L’algoritmo

Vincenzo Della Mea e Stefano Mizzaro

Università di Udine
dellamea@dimi.uniud.it mizzaro@dimi.uniud.it
http://www.dimi.uniud.it/~dellamea
http://www.dimi.uniud.it/~mizzaro

Premessa

In un corso di introduzione all'informatica è indispensabile fornire competenze tecniche, per consentire agli studenti di utilizzare in modo proficuo, a qualche livello, un calcolatore elettronico. È però importante fornire anche nozioni concettuali che arricchiscano la cultura degli studenti, non solo come principio educativo generale, ma anche perché, a causa del rapido sviluppo tecnologico, le conoscenze e capacità tecniche degli utilizzatori di calcolatori diventano spesso obsolete. Queste nozioni culturali, che sono più assestate, permettono di adeguarsi alle innovazioni tecnologiche.

In queste note vogliamo illustrare in modo intuitivo ma rigoroso il concetto di algoritmo, nella convinzione che in un corso introduttivo all'informatica sia importante evidenziare che il calcolatore elettronico è solo uno strumento, un esecutore di algoritmi, ma che si può "fare informatica" anche senza calcolatori elettronici. Bisogna quindi introdurre il concetto di algoritmo *prima* di parlare di un calcolatore elettronico, ma evitando formalismi troppo complicati, o matematicamente sofisticati, o di scarsa espressività (ad esempio le macchine di Turing o le funzioni parziali ricorsive). Per fare ciò abbiamo modificato la *metafora dell'ufficio*, presentata in [Gui96] e ispirata alla *stanza cinese* di Searle [Sea90] (illustrata nel cap. 7).

Queste note sono state pensate e realizzate per corsi introduttivi all'informatica di livello universitario in corsi di laurea e diploma. Quindi la matematica adottata è molto semplice e non è presupposta alcuna conoscenza della programmazione. Sono state adottate come dispense dei corsi seguenti all'Università di Udine:

- Facoltà di Agraria, Corso di Laurea in Scienze e Tecnologie Alimentari, Modulo di Laboratorio di Informatica;
- Facoltà di Agraria, Corso di Laurea in Scienze e Tecnologie Agrarie, Modulo di Informatica Applicata;
- Facoltà di Agraria, Corso di Diploma in Tecnologie Alimentari, Modulo di Informatica;
- Facoltà di Lettere e Filosofia, Corso di Diploma di Operatore dei Beni Culturali, insegnamento di Informatica Generale;
- Facoltà di Medicina e Chirurgia, Corso di Diploma di Tecnico Sanitario di Laboratorio Biomedico, Modulo di Informatica;
- Facoltà di Ingegneria, Corso di Diploma in Ingegneria Elettronica, Modulo di Fondamenti di Informatica.

Al termine di ogni capitolo sono proposti alcuni esercizi, e in alcuni capitoli sono presenti riferimenti bibliografici che possono essere usati per approfondire alcuni argomenti e/o proporre tesine agli studenti.

Indice

Capitolo 1.
Informatica = informazione + algoritmo

«*L'informatica è la scienza degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione.*» Questa è la definizione di informatica proposta dall'ACM (Association for Computing Machinery), la principale organizzazione che riunisce informatici di tutto il mondo. Due sono i concetti fondamentali di questa definizione: l'informazione e l'algoritmo. Tradizionalmente, nei corsi di informatica, si pone maggiore attenzione sugli algoritmi, e noi non cambieremo questa impostazione. Incominciamo però spendendo qualche parola sull'affascinante concetto di informazione. Parleremo poi brevemente dell'algoritmo e infine ritorneremo sull'informatica e sulla definizione dell'ACM.

1.1 L'era dell'informazione

Viviamo nell'*era dell'informazione*. Negli ultimi decenni la quantità e la varietà di informazioni disponibili sono enormemente aumentate rispetto al passato, come ognuno di noi può constatare. Spesso si è portati a credere che la centralità dell'informazione sia un fenomeno solo tecnologico, tipico degli ultimi decenni, ma non è così. Per rendersene conto bisogna adottare una visione più generale e collocarsi in un quadro storico più ampio, prendendo in considerazione avvenimenti che risalgono a qualche decennio fa.

L'informazione è un argomento che da sempre ha suscitato l'interesse di persone appartenenti ai campi più disparati: filosofi, artisti, umanisti. La scienza, dedicandosi allo studio di materia ed energia, ha trascurato per secoli il concetto di informazione ma, a partire all'incirca dalla metà di questo secolo, l'attenzione del mondo scientifico su questo argomento è aumentata notevolmente. Scienziati quali Gödel, Turing, Shannon, Wiener, von Bertalanffy, e altri, hanno effettuato importanti scoperte nel settore della logica matematica e hanno fondato le discipline dell'informatica, della teoria dell'informazione, della cibernetica, della teoria dei sistemi [Göd31, Tur36, Sha48, SW49, Wie48, vB68]. A partire da quegli anni, il concetto di informazione ha cominciato ad assumere un assetto più formale e sono stati costruiti i primi calcolatori elettronici in grado di memorizzare e gestire grandi quantità di dati, informazioni e conoscenze.

Ma soprattutto ci si è resi conto che, accanto al mondo della materia, delle forze e dell'energia (il *Pleroma*, adottando la terminologia cara a Bateson [Bat84]), esiste un mondo dell'informazione, della conoscenza, del significato (la *Creatura*) in cui

le ben note leggi della fisica sembrano sorprendentemente non valere: le leggi di conservazione, valide per energia e quantità di moto, non sussistono per l'informazione; l'assenza di informazione può essere informazione; nella Creatura, qualsiasi cosa può rappresentare qualsiasi altra cosa; e così via.

La Creatura sembra sfuggire alle capacità di comprensione e descrizione della scienza tradizionale (la fisica) ben più di quanto faccia il mondo fisico: la teoria dell'informazione di Shannon, probabilmente il contributo teorico che riesce a catturare in modo più completo il concetto di informazione, riesce a trattare in modo soddisfacente solo l'aspetto sintattico dell'informazione, e trascura semantica (significato) e pragmatica (uso) [Lon98]. La *scienza dell'informazione*[1] sembra trovarsi ancora in uno stato primordiale, ben lontana da discipline quali la meccanica, o la fluidodinamica, in cui a partire dalle definizioni formali di pochi semplici concetti si riescono a dedurre leggi fondamentali che permettono di descrivere e prevedere parecchi fenomeni. Vi sono due posizioni al riguardo.

Da un lato c'è chi ritiene che sia solo questione di tempo: in fondo la scienza dell'informazione è nata circa cinquanta anni fa, mentre le ultime rivoluzionarie scoperte della fisica, la relatività e la meccanica quantistica, hanno ormai ben più di cinquant'anni, per non parlare della fisica classica, vecchia di secoli. Inoltre, nel settore informatico, la tecnica sembra aver preso il sopravvento, relegando la cultura e la riflessione critica a un ruolo di secondo piano, facilitando la diffusione di un gergo incomprensibile ai non addetti ai lavori e ostacolando la crescita di una nuova scienza.

Dall'altro lato c'è chi assume una posizione più radicale, ma anch'essa giustificata, sostenendo che le leggi del Pleroma non sono trasferibili alla Creatura e che il problema è costituito dalla mancanza di strumenti di base adeguati per comprendere e descrivere l'universo dell'informazione. Quando Galileo Galilei ha ipotizzato che il colore di un grave che cade non è "importante" per comprenderne il moto e che quindi è possibile astrarre dalle caratteristiche non pertinenti per comprendere e descrivere un fenomeno fisico, e che tale descrizione può essere effettuata in linguaggio matematico, ha fatto due ipotesi fondamentali, sulle quali si è basata la scienza moderna. Tali ipotesi sono sembrate corrette per alcuni secoli, ma i paradossi della meccanica quantistica [Pen92] e del caos [Pri93] hanno cominciato a farle vacillare. La scienza dell'informazione potrebbe dare loro il colpo di grazia: forse, come sostenuto a esempio in [Dev91], non abbiamo a disposizione la matematica adatta per modellare l'informazione.

Non intendiamo qui prendere posizione su tali questioni (è probabilmente prematuro farlo: forse solo il tempo ci dirà chi è nel giusto). Quello che è importante notare è che molto resta ancora da fare, che manca una teoria dell'informazione completa e che le radici dell'odierna *società dell'informazione* vanno collocate quantomeno nell'immediato secondo dopoguerra, quando si è manifestato l'interesse del mondo scientifico per l'informazione. Come talora accade, la tecnologia ha recepito le scoperte scientifiche con un ritardo di una ventina d'anni, ma il ruolo della tecnologia (radio, televisione, calcolatori elettronici, reti), seppure successivo, non è stato senz'altro secondario rispetto a quello della scienza. È infatti grazie all'enorme diffusione dei calcolatori, alle connessioni via rete, all'utilizzo di supporti ad alta capacità quali i CD-ROM, che oggi si sentono sempre più spesso espressioni quali "problema del sovraccarico di

[1] La locuzione "scienza dell'informazione" è qui da interpretare in modo letterale, come la scienza che studia l'informazione, e non come la scienza dei calcolatori, a cui ci riferiamo con "informatica".

informazioni", "troppe informazioni = nessuna informazione", "esplosione dell'informazione", "diluvio informazionale" o addirittura "caos informazionale".

Negli ultimi 5-10 anni, la crescita esponenziale del fenomeno *Internet* (oramai siamo a più di 100 milioni di utenti in tutto il mondo) ha accentuato questa situazione: ognuno di noi, disponendo di un calcolatore collegato in Internet, ha libero accesso a una banca dati multimediale, distribuita su tutto il globo, di proporzioni inimmaginabili fino a poco tempo fa, costituita dai dati messi a disposizione sui calcolatori collegati in rete. Su questa banca dati possiamo inserire i nostri dati, "navigare" e ricercare informazioni, con gli obiettivi più diversi (lavoro, studio, divertimento).

Solo il futuro ci dirà se prevarranno le visioni ottimistiche, come quella di Negroponte [Neg95], o se invece si avvereranno gli scenari più pessimistici, quale quello ipotizzato da Postman [Pos93]. Negroponte ritiene che arriveremo a realizzare una rete planetaria in cui si potranno trasportare non gli atomi, ingombranti e costosi, ma i *bit*, leggeri ed economici, e che i calcolatori riusciranno a salire di livello, a costruire un modello dell'utente, a capire quali informazioni gli interessano e a selezionarle in modo autonomo e automatico. Secondo Postman corriamo invece il serio pericolo di restare vittime di una sorta di AIDS (Anti-Information Deficiency Syndrome): per riuscire a gestire tutti i dati disponibili avremo bisogno di più tecnologia informatica, ciò consentirà alla quantità di dati di lievitare, richiedendo ancora più tecnologia, e così via, in un circolo vizioso che porterà al collasso culturale.

Insomma, scienziati, tecnici e sociologi concordano che, grazie a scoperte di natura sia scientifica sia tecnologica, ci troviamo in questi anni nel bel mezzo di una vera e propria *rivoluzione informazionale*, che potrebbe rivelarsi di importanza paragonabile alla rivoluzione industriale dei secoli scorsi.

1.2 L'informazione

Ma che cos'è l'informazione? È difficile rispondere a questa domanda, anche perché il termine "informazione", nell'uso quotidiano, ha almeno due significati. Da un lato spesso diciamo "c'è molta informazione in quel libro", assumendo implicitamente una visione *oggettiva* dell'informazione, una visione che ritiene l'informazione immanente, intrinseca nel suo supporto o contenitore. Dall'altro lato, spesso diciamo "questo libro non mi dà nessuna informazione", con un'accezione *soggettiva* del termine. Un lungo testo scritto in giapponese *contiene* più informazione di un testo più corto, ma un testo giapponese non ci *dà* nessuna informazione (se non conosciamo il giapponese).

Una proposta è quella di Bateson, che vede l'informazione come una *differenza*: «Ricevere informazioni vuol dire sempre e necessariamente ricevere notizie di *differenza*.» [Bat84, pg. 96]. Una luce accesa è differente da una luce spenta; un "sì" è diverso da un "no", un "1" da uno "0", la "presenza" dall'"assenza" (e quindi l'assenza di informazione può essere informazione perché diversa dalla presenza di informazione), ecc. ecc.

Il già citato Shannon ha proposto nel 1948 un'elegante teoria matematica che cattura in modo elegante la *quantità* di informazione [Sha48]. La quantità di informazione che un qualsiasi evento può portare dipende dal numero di alternative e dalla loro probabilità. Ad esempio, se noi telefoniamo a un nostro amico australiano e gli chiediamo "Piove?", possiamo ottenere due risposte: "sì" o "no". Se noi gli poniamo 2 domande anziché una sola, il numero di risposte

possibili aumenta. Ad esempio, supponiamo di chiedergli “Piove?” e “Tua moglie è a casa?”; le combinazioni di risposte possibili saranno “sì/sì”, “sì/no”, “no/sì”, “no/no”. È intuitivo che con due domande possiamo ottenere più informazioni: le alternative possibili sono in numero maggiore. Se poniamo 3 domande (sempre formulate in modo tale che le risposte siano esclusivamente “sì” o “no”), avremo 8 possibili combinazioni; in generale, se poniamo N domande avremo 2^N possibilità.

La quantità di informazione dipende quindi dal numero di alternative. L’unità di misura della quantità di informazione è il *bit*: con “Piove?” otteniamo 1 bit di informazione; con 2 domande 2 bit; in generale, con N domande otteniamo N bit.

Supponiamo ora che all’altro capo del nostro telefono vi sia un nostro amico di Londra, che la telefonata venga fatta durante il mese di novembre e che, come in precedenza, gli chiediamo “Piove?”. Quasi sicuramente la risposta sarà “sì” (magari seguita da una serie di considerazioni più o meno colorite sul tempo metereologico londinese). Ma questa risposta non ci porta molte informazioni: già prima di sentirla siamo quasi certi che la risposta sarà “sì”. Bene, qui entra in gioco il concetto di *probabilità*: se un evento è molto probabile, porta poca informazione (a Londra è molto probabile che piova; il fatto che effettivamente piova non ci dice molto); se un evento è poco probabile, porta più informazione (saremmo sorpresi se a Londra ci fosse il sole). Agli estremi, se un evento è certo (probabilità 1, ossia massima) l’informazione è nulla (ad esempio, se qualcuno ci dice “Uno più uno fa due”, ci fornisce ben poca informazione); se un evento è impossibile, l’informazione portata è infinita.

Tutto ciò è ragionevole: se noi sappiamo già che Pasquale ha 12 anni, la frase “Pasquale ha 12 anni” non ci dà nessuna informazione. Se però qualcuno ci dice che “Pasquale ha 13 anni” e ci mostra il suo certificato di nascita per convincerci, allora l’informazione ricevuta è molta di più (la nostra conoscenza varia parecchio). Asintoticamente, se noi siamo assolutamente certi che Pasquale ha 12 anni, l’informazione ricevuta è infinita.

Si osservi per inciso che finora abbiamo ipotizzato che le domande siano *indipendenti* l’una dall’altra: se chiedessimo “Tua moglie è a casa?” e “Tua moglie è uscita?” ovviamente avremmo solo due possibili alternative, “sì/no” e “no/sì”, e non quattro. Questo dipende dal fatto che i due eventi “essere a casa” e “essere fuori di casa” dipendono l’uno dall’altro (sono incompatibili, uno impedisce l’altro). Ovviamente vi possono essere anche altri legami fra eventi: un evento ne può implicare un altro (dopo “Tua moglie è andata a fare la spesa?” è inutile chiedere “Tua moglie è uscita?”); un evento può rendere più o meno probabile un altro (si rifletta su “Tua moglie è andata a fare la spesa?” e “Avete la dispensa strapiena?”), ecc.

Con il diffondersi di Internet e del World Wide Web la quantità di informazioni a disposizione di ogni utente di un calcolatore collegato in rete è senza dubbio fortemente aumentata. A questo aumento quantitativo non si è però affiancato un miglioramento qualitativo: oggi tutti possono immettere in rete i propri documenti, le proprie idee. Ma spesso quando tutti parlano, pochi ascoltano, e quasi nessuno riflette... Questo è forse il problema principale da affrontare oggi: su Internet si trova di tutto (figura 1.1) ma la qualità è spesso scadente.

Figura 1.1 - Su Internet si trova di tutto...

1.3 L'algoritmo

Il concetto di algoritmo verrà definito in modo più preciso nei prossimi capitoli. Per ora ci limitiamo a dire che un algoritmo è un procedimento, composto da una sequenza di istruzioni elementari, che consente di rispondere a un insieme di domande, o di risolvere un problema. Esistono algoritmi per calcolare il prodotto di due numeri di più cifre, per trovare il massimo di un insieme di numeri, per ordinare alfabeticamente una serie di nomi, ecc.

Gli algoritmi sono studiati in modo formale all'interno della teoria della calcolabilità, utilizzando strumenti matematici troppo sofisticati per essere presentati in questa sede. Nei prossimi capitoli proponiamo una metafora che illustra in modo intuitivo ma rigoroso questo concetto.

1.4 L'informatica

L'informatica non è quindi né la "scienza di calcolatori elettronici" né la "scienza dell'informazione". Queste due definizioni sono piuttosto diffuse fra chi non frequenta l'ambiente informatico, e sembrano sostenibili: in inglese il termine informatica è tradotto con *computer science*; in Italia la laurea in informatica era denominata fino a qualche anno fa "Scienze dell'informazione". Però la prima definizione è troppo restrittiva: il termine calcolatore elettronico, o *computer*, è solo uno strumento (si può fare informatica senza calcolatore) e non appare nella definizione dell'ACM. La seconda è invece troppo generale (comprenderebbe ad esempio il giornalismo!).

Inoltre, possiamo osservare come l'informatica abbia portato a due innovazioni. La prima è di natura quantitativa e tecnologica, ed è legata alla quantità di informazione: grazie alla diffusione dei calcolatori, dei supporti di memorizzazione di dati e delle reti, la quantità di informazione disponibile per ciascuno di noi è enormemente aumentata. La seconda innovazione è di natura qualitativa e concettuale, ed è legata ai concetti di informazione e algoritmo, concetti nuovi che hanno portato alla nascita di una nuova scienza a metà strada

fra la matematica e l'ingegneria e hanno consentito di gestire le enormi quantità di dati.

L'informatica è in realtà una disciplina articolata e in rapida evoluzione. È possibile individuare numerose sottodiscipline, e questa suddivisione può essere effettuata in molti modi. Una proposta è la seguente:

- Algoritmi e strutture dati (studio di algoritmi più efficaci).
- Programmazione (studio dei programmi, ossia degli algoritmi espressi in un linguaggio comprensibile dal calcolatore; studio di tali linguaggi, detti linguaggi di programmazione).
- Architettura dei calcolatori (come costruire calcolatori più potenti).
- Reti di calcolatori (come collegare più calcolatori).
- Sistemi operativi (studio dei programmi che consentono di utilizzare i calcolatori).
- Ingegneria del software (costruzione di programmi di grandi dimensioni).
- Basi di dati e sistemi per il reperimento delle informazioni (programmi in grado di gestire grandi quantità di dati)
- Intelligenza artificiale (programmi in grado di imitare il comportamento intelligente degli esseri umani)
- Visione e robotica (programmi in grado di "vedere" e programmi con un corpo)
- Teoria dell'informazione (studio teorico della rappresentazione efficace dell'informazione)

La diffusione dei calcolatori, e delle banche dati (depositi di dati di enormi quantità) ha importanti conseguenze sociali: se si sa che un'informazione si può trovare in una banca dati, si è meno stimolati a ricordarla a memoria. Questo fenomeno, denominato *estroflessione cognitiva* in [Lon98] (a significare che alcune capacità cognitive sono "portate all'esterno", al di fuori della mente), può essere giudicato un fenomeno pericoloso o un semplice cambiamento: è vero che non si imparano più le poesie a memoria, ma anche l'introduzione della scrittura è stato un fenomeno analogo.

Nel seguito di queste note descriviamo la metafora dell'ufficio e i diagrammi di flusso (cap. 2); essi sono adottati per presentare il concetto di algoritmo (cap. 3), per introdurre la teoria della calcolabilità, che studia le proprietà formali degli algoritmi (cap. 4) e per discutere le nozioni di problema e soluzione (cap. 5); nel cap. 6 vengono introdotte alcune restrizioni ai diagrammi di flusso, in modo da renderli più simili ai linguaggi di programmazione moderni; infine, nel cap. 7 vengono analizzati alcuni problemi fondazionali sollevati dal concetto di algoritmo.

Capitolo 2. La metafora dell'ufficio

Comprendere i concetti di (e i rapporti tra) utente, calcolatore, programmatore e programma non è sempre immediato. A questo scopo, può risultare utile il ricorso a una metafora, che illustri questi concetti in termini più vicini alle conoscenze quotidiane dell'utente. La metafora scelta è quella dell'ufficio, secondo quanto proposto da Guida [Gui96], ma con alcune modifiche per renderla ancora più semplice e intuitiva, pur mantenendone il rigore formale.

In questo capitolo vengono descritti l'ufficio e il suo funzionamento, le modalità di descrizione delle attività dell'impiegato con il formalismo dei diagrammi di flusso, e alcuni esempi di funzionamento dell'ufficio stesso.

2.1 L'ufficio

L'ufficio è costituito da una stanza singola, in cui si trova un impiegato che agisce sotto la guida di un direttore. L'ufficio comunica verso l'esterno con uno sportello per l'utente. La dotazione dell'ufficio comprende:

- Una sedia, su cui l'impiegato sta seduto quando non ha niente da fare.
- Una *lavagna di lavoro*, a quadretti, costituita da M righe, che chiameremo *posizioni*, suddivise in N *caselle*. Ogni posizione può contenere un numero naturale costituito da N cifre: questo significa che la lavagna può contenere M numeri naturali, ognuno dei quali, avendo al massimo N cifre, sarà compreso tra 0 e 10^N - 1. La *cornice* di questa lavagna è particolare, in quanto sul lato sinistro assume una larghezza corrispondente a 5 quadretti, ed è suddivisa in righe esattamente corrispondenti alle righe della lavagna stessa. Questa cornice ospiterà dei *nomi*, composti al più da 5 caratteri (di cui il primo dovrà essere alfabetico). Ogni nome dovrà essere univoco (diverso dagli altri) e si suppone sia il nome della corrispondente posizione sulla lavagna. Chiamiamo *variabili* le posizioni sulla lavagna, identificate ognuna da un nome distinto, e *valore* il loro eventuale contenuto. La figura 2.1 illustra graficamente la lavagna di lavoro (con N = 8 e M = 10).

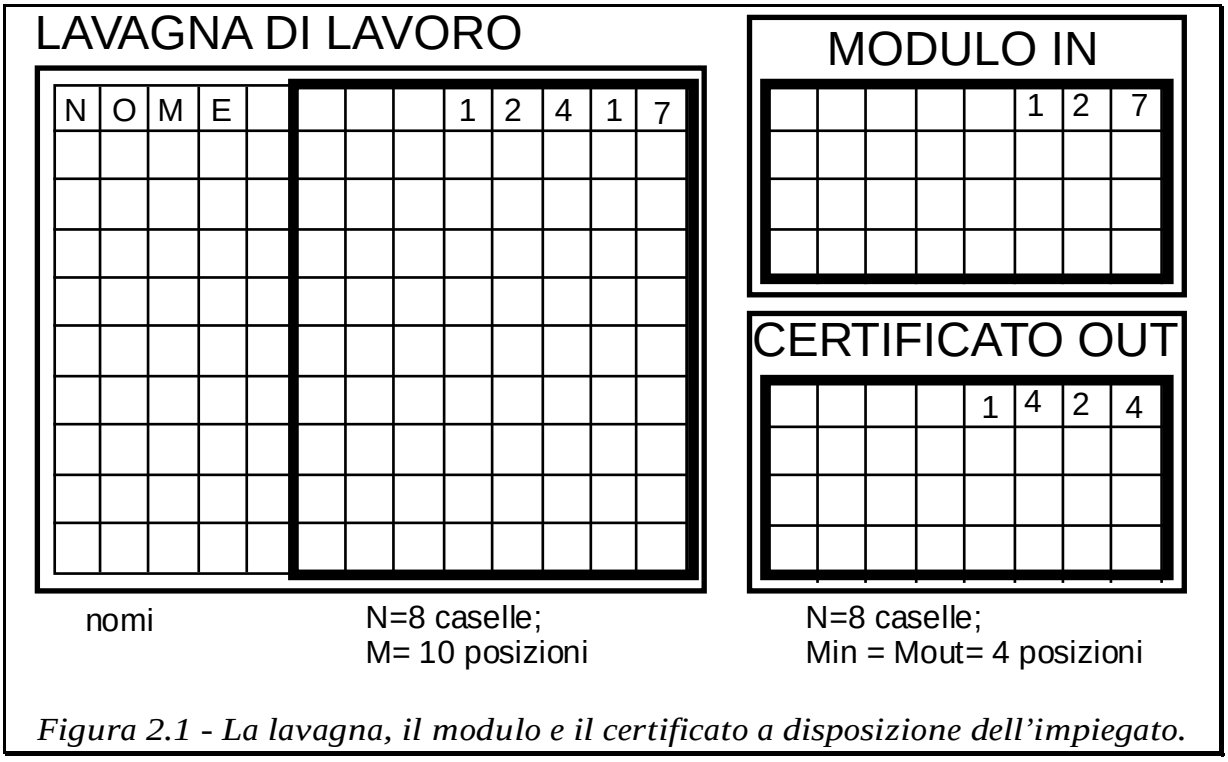

Figura 2.1 - La lavagna, il modulo e il certificato a disposizione dell'impiegato.

- Nello sportello utente sono presenti due mezzi per comunicare con l'esterno: un modulo di ricezione di dati dall'utente, chiamato *modulo IN*, e un certificato per la comunicazione di risultati all'utente, denominato *certificato OUT*. Sia il modulo sia il certificato sono organizzati in posizioni (per un totale rispettivamente M_{IN} e M_{OUT}) di N caselle come la lavagna di lavoro, ma non hanno cornice, in quanto si suppone che le loro posizioni abbiano nomi standard IN-i e OUT-j, con i compreso tra 1 e M_{IN} e j compreso tra 1 e M_{OUT}. La compilazione del primo è a cura dell'utente, quella del secondo è compito dell'impiegato. Anche modulo e certificato sono presentati in figura 2.1 (con N = 8 e M_{IN} e M_{OUT} supposti uguali a 4).
- Un *gesso* e un *cancellino*: vengono usati dall'impiegato per scrivere e cancellare sulla lavagna di lavoro, inclusa la cornice. Prima di scrivere in una posizione, l'impiegato cancella ciò che c'era scritto prima.
- Una lavagna magnetica, detta *lavagna di programma*. Su questa lavagna viene descritto, secondo un'opportuna modalità che vedremo, il compito che l'impiegato deve svolgere, in base a una serie di attività elementari che egli è in grado di eseguire. Il compito viene descritto sulla lavagna da un *direttore* dell'ufficio.
- Un *segnalino magnetico*, a forma di freccia, da posizionare sulla lavagna magnetica in corrispondenza dell'attività elementare che l'impiegato sta per svolgere.
- Un *segnalatore acustico* collegato con un pulsante messo all'esterno dell'ufficio.

L'aspetto globale dell'uffico è sintetizzato graficamente in figura 2.2.

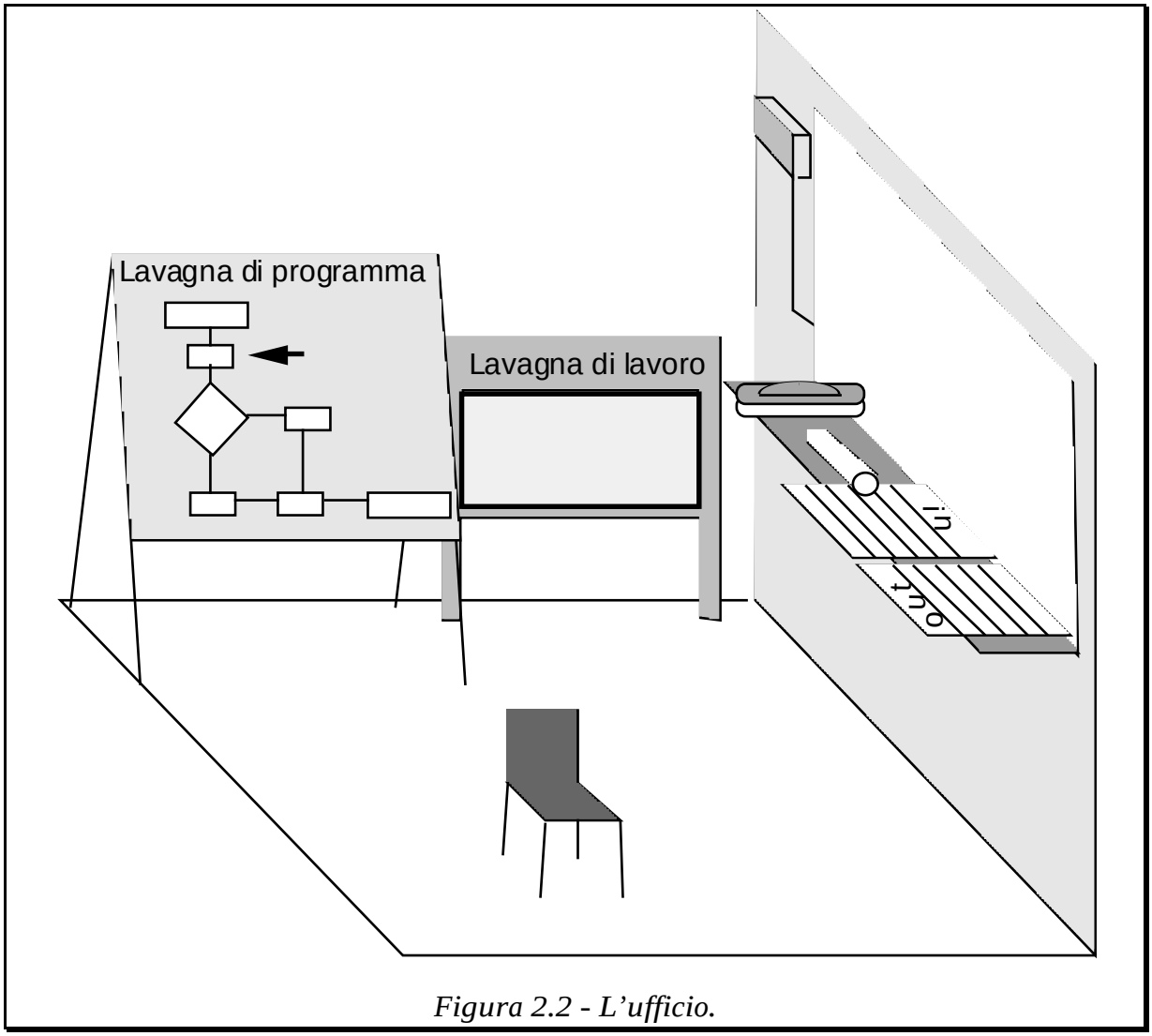

Figura 2.2 - L'ufficio.

2.2 Il funzionamento dell'ufficio

L'attività dell'impiegato si esplica attraverso i passi seguenti:

1. L'impiegato attende seduto sulla sua sedia che suoni il segnale acustico collegato al pulsante esterno.
2. Quando il segnalatore suona, si alza, si reca alla lavagna di lavoro e la cancella, poi va alla lavagna di programma e posiziona il segnalino magnetico nel punto della descrizione contrassegnato come inizio della sua attività.
3. A quel punto, inizia a seguire le istruzioni descritte sulla lavagna di programma, eseguendo una ad una le attività elementari nell'ordine stabilito. L'impiegato può leggere dal modulo IN eventuali dati, utilizzare la lavagna di lavoro per memorizzare eventuali dati temporanei, spostare il segnalino magnetico all'istruzione successiva e riportare i risultati finali sul certificato OUT.

L'utente può richiedere all'impiegato di effettuare l'attività indicata sulla lavagna di lavoro semplicemente premendo il pulsante collegato al segnalatore

acustico. Prima di fare ciò, avrà cura di riportare sul modulo IN i dati che vuole far elaborare dall'impiegato, e al termine leggerà sul certificato OUT i dati frutto dell'elaborazione.

Quando interviene il direttore? Egli è in grado di conoscere le esigenze degli utenti, di identificare i problemi che essi hanno, e di trovare metodi di soluzione appropriati per risolverli. Una volta trovato il metodo, è in grado di progettare una descrizione operativa per portarlo a termine, in una modalità comprensibile dall'impiegato, che la leggerà dalla lavagna di programma dove il direttore l'ha rappresentata. In altre parole, su richiesta degli utenti il direttore fornirà all'impiegato le istruzioni per eseguire il procedimento che risolve un particolare problema, ovvero *l'algoritmo* per la sua risoluzione.

2.3 Le attività dell'impiegato

L'impiegato è in grado di eseguire un insieme finito di attività elementari. Inoltre può eseguirle in sequenza, oppure ripetutamente, oppure scegliendo tra attività alternative. Le attività elementari e il modo in cui esse si combinano per formare un'attività più complessa vengono specificati sulla lavagna di programma utilizzando una metodologia che l'impiegato comprende, e in cui rientrano due categorie di "istruzioni": quelle che indicano un'attività elementare e quelle che indicano come organizzare queste attività. I prossimi due sottoparagrafi si occupano di queste due categorie.

2.3.1 Le attività elementari

Le attività elementari dell'impiegato includono operazioni di lettura e scrittura sulle lavagne a sua disposizione (l'impiegato non può scrivere sulla lavagna di programma), e operazioni aritmetiche. In particolare abbiamo:

- Istruzione di assegnamento:

  $$valore \rightarrow nome$$
  $$nome1 \rightarrow nome2$$

  La prima forma dell'istruzione di assegnamento scrive il numero specificato da *valore* nella posizione della lavagna di lavoro specificata da *nome*, dopo aver cancellato l'eventuale contenuto precedente. La seconda forma dell'istruzione di assegnamento copia il contenuto della posizione specificata da *nome1* nella posizione specificata da *nome2*. Se sulla lavagna non esistono posizioni contrassegnate da *nome* (nel primo caso) o *nome2* (nel secondo caso), l'impiegato provvede a scrivere tale nome sulla cornice nella prima posizione libera sulla lavagna partendo dall'alto. Questa operazione è detta *allocazione*.
- Istruzione di lettura:

  LEGGI *IN-i nome*
  LEGGI *nome1 nome2*

  Con la prima forma dell'istruzione di lettura, il numero naturale contenuto nella posizione *IN-i* del modulo IN viene letto e scritto nella posizione contrassegnata da *nome* sulla lavagna di lavoro. Si suppone che nella posizione *IN-i* sia effettivamente presente un numero, e che *i* sia minore o uguale a M_{in}. La seconda forma è analoga: il valore contenuto in *nome1* indica da quale posizione del modulo IN leggere il valore da scrivere nella posizione contrassegnata da *nome2*.

- Istruzione di scrittura:

  SCRIVI *nome OUT-i*
  SCRIVI *nome1 nome2*

  Anche qui abbiamo due possibili forme. Nella prima, il numero contenuto nella posizione *nome* della lavagna di lavoro viene scritto sul certificato OUT nella posizione *OUT-i*. Si suppone che *i* sia minore o uguale a M_{out}. Nella seconda il numero contenuto nella posizione *nome1* della lavagna di lavoro viene scritto sul certificato OUT nella posizione indicata da *nome2*.
- Operazioni aritmetiche:

  op1 + *op 2* → *nome*
  op 1 - *op 2* → *nome*
  op 1 · *op 2* → *nome*
  op 1 / *op 2* → *nome*

  L'impiegato è in grado di effettuare somma, sottrazione, prodotto, e divisione intera. *op1* e *op2* possono essere sia nomi relativi a posizioni sulla lavagna di lavoro sia numerali, ovvero direttamente numeri naturali. Nel caso siano nomi di posizioni, viene dapprima letto il valore corrispondente. Con i valori degli operandi viene calcolato il valore corrispondente all'operazione definita dall'istruzione, e il risultato viene scritto nella posizione contrassegnata da *nome*. Se *nome* non è presente sulla lavagna di lavoro, allora lo scrive sulla cornice nella prima posizione libera prima di scrivere il valore sulla lavagna.

L'impiegato capirà quindi istruzioni del tipo:

- 4 → *ALFA* (scrive il valore 4 nella posizione *ALFA*, se esiste; altrimenti prima scrive anche il nome ALFA nella prima posizione libera; questo vale per ogni esempio successivo)
- *BETA* → *ALFA* (legge il valore nella posizione *BETA* e lo scrive nella posizione *ALFA*)
- *GAMMA* - *DELTA* → *FI* (legge i valori contenuti in *GAMMA* e *DELTA*, sottrae il secondo dal primo, e scrive il risultato in *FI*)
- *MESE* + *1* → *MESE* (incrementa di 1 il valore contenuto in *MESE*)
- LEGGI *IN-3 RO* (legge il valore dalla terza posizione sul modulo IN e lo scrive sulla lavagna di lavoro alla posizione *RO*)
- SCRIVI *KAPPA OUT-5* (legge il valore dalla posizione *KAPPA* sulla lavagna di lavoro e lo scrive sul certificato OUT nella quinta posizione)

Inoltre, l'impiegato è in grado di valutare condizioni logiche relative ai valori memorizzati nelle posizioni della lavagna di lavoro e a numeri, calcolando il valore di verità di un'espressione costituita da semplici confronti tra numeri (>, <, =) eventualmente combinati attraverso connettivi logici (AND, OR, NOT).

Più precisamente, l'impiegato può valutare i seguenti tipi di espressioni logiche semplici:

op1 > *op2* (vera se il valore di *op1* è maggiore del valore di *op2*)
op1 < *op2* (vera se il valore di *op1* è minore del valore di *op2*)
op1 = *op2* (vera se il valore di *op1* è uguale al valore di *op2*)

dove *op1* e *op2* possono essere sia nomi relativi a posizioni sulla lavagna di lavoro sia numerali.

Le espressioni possono essere combinate come segue:

(espr1) AND (*espr2*) (vera se entrambe le espressioni *espr1* ed *espr2* sono vere)

(espr1) OR (*espr2*) (vera se almeno una fra *espr1* ed *espr2* è vera)
NOT (*espr1*) (vera se *espr1* è falsa)

Le parentesi possono essere omesse se non c'è ambiguità.
L'impiegato saprà quindi valutare condizioni logiche del tipo:

- *ALFA* > 1 (vera se il valore contenuto nella posizione *ALFA* è maggiore di 1)
- *BETA* = *ALFA* (vera se il contenuto della posizione *BETA* è uguale al contenuto della posizione *ALFA*)
- (*GAMMA* < 1) AND (*DELTA* = *FI*) (vera se il valore di *GAMMA* è minore di 1, e contemporaneamente il valore di *DELTA* è uguale al valore di *FI*)
- NOT (*GAMMA* > 5) (vera se *GAMMA* è minore o uguale a 5)
- ((*ALFA* < 87) OR (*BETA* = 18)) AND (NOT (*GAMMA* = 0)) (vera se il valore di *GAMMA* è diverso da 0, e contemporaneamente si ha che *ALFA* è minore di 87, oppure *BETA* è uguale a 18, oppure entrambe le condizioni assieme).
- (*ALFA* < 87 OR *BETA* = 18) AND (NOT *GAMMA* = 0) (stesso valore di verità della precedente, ma con meno parentesi).

2.3.2 L'organizzazione delle attività: i diagrammi di flusso

L'organizzazione delle attività elementari dell'impiegato in attività più complesse viene descritta dal direttore sulla lavagna di programma utilizzando i *diagrammi di flusso*.

Un diagramma di flusso è una struttura costituita da blocchi connessi da frecce (più formalmente, sono grafi costituiti da nodi e archi orientati che collegano coppie di nodi). Esistono quattro tipi di blocchi:

- un *blocco di inizio*, che indica l'inizio del programma; è unico in un programma, e può presentare un solo arco uscente e nessuno entrante;
- un *blocco di fine*, che denota la fine del programma e quindi non avrà archi uscenti, ma solo uno o più archi entranti; possono essercene più d'uno;
- *blocchi funzionali*, che contengono un'istruzione elementare di tipo aritmetico, di assegnamento, di lettura e scrittura, rappresentata come descritto nel paragrafo precedente; possono avere più archi entranti, ma solo uno uscente;
- *blocchi decisionali*: possono contenere una condizione logica che assume i valori vero (V) oppure falso (F); possono avere più archi entranti, e da esso ne escono due, uno denotato con V, e l'altro con F.

La figura 2.3 indica come vengono rappresentati i quattro tipi di blocchi.

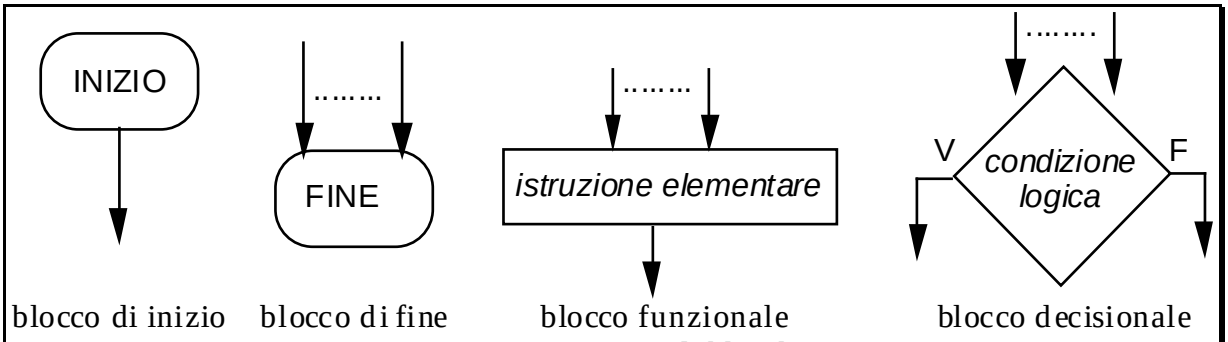

Figura 2.3 - I tipi di blocchi.

Un diagramma di flusso può essere costituito da un numero qualsiasi di blocchi, purché ce ne sia uno e uno solo di inizio, e almeno uno di fine.

Limpiegato riconosce sulla lavagna i vari tipi di blocchi, e li interpreta come segue:

- Blocco di inizio: quando il segnale acustico suona, l'impiegato cerca il blocco di inizio sulla lavagna e come prima cosa posiziona il segnalino magnetico accanto ad esso. Subito dopo identifica qual è il blocco connesso a quello di inizio tramite la freccia, e posiziona accanto a quello il segnalino magnetico.
- Blocco di fine: quando il segnalino magnetico finisce accanto al blocco di fine, l'impiegato torna a sedersi sulla sedia.
- Blocco funzionale: l'impiegato esegue l'attività elementare descritta all'interno del blocco, utilizzando eventualmente la lavagna di lavoro e leggendo dati sul modulo IN o scrivendo dati sul certificato OUT. Al termine dell'esecuzione, identifica il blocco connesso tramite la freccia a quello appena eseguito, e posiziona accanto ad esso il segnalino magnetico.
- Blocco decisionale: l'impiegato valuta la condizione logica contenuta nel blocco, e in base al risultato (vero o falso) posiziona il segnalino magnetico al blocco connesso alla freccia etichettata con V o con F.

2.4 Esempio di funzionamento

Vediamo ora il funzionamento dell'ufficio in un caso concreto.

Un utente si reca dal direttore e gli presenta il suo problema: vuole sapere quanto vale la terza potenza di 4. Il direttore gli risponderà che studierà il problema e appena avrà pronta la soluzione lo avvertirà.

Il direttore analizza quanto l'utente gli ha chiesto, identifica un metodo matematico per il calcolo delle potenze (la moltiplicazione di un numero ripetuta per il numero di volte indicato dall'esponente, a meno che l'esponente sia zero, nel qual caso il risultato è sempre 1) e lo traduce nel diagramma di flusso corrispondente, basandosi sulle istruzioni elementari che l'impiegato conosce (tra le quali non c'è la potenza). Il risultato è il diagramma di flusso di figura 2.4.

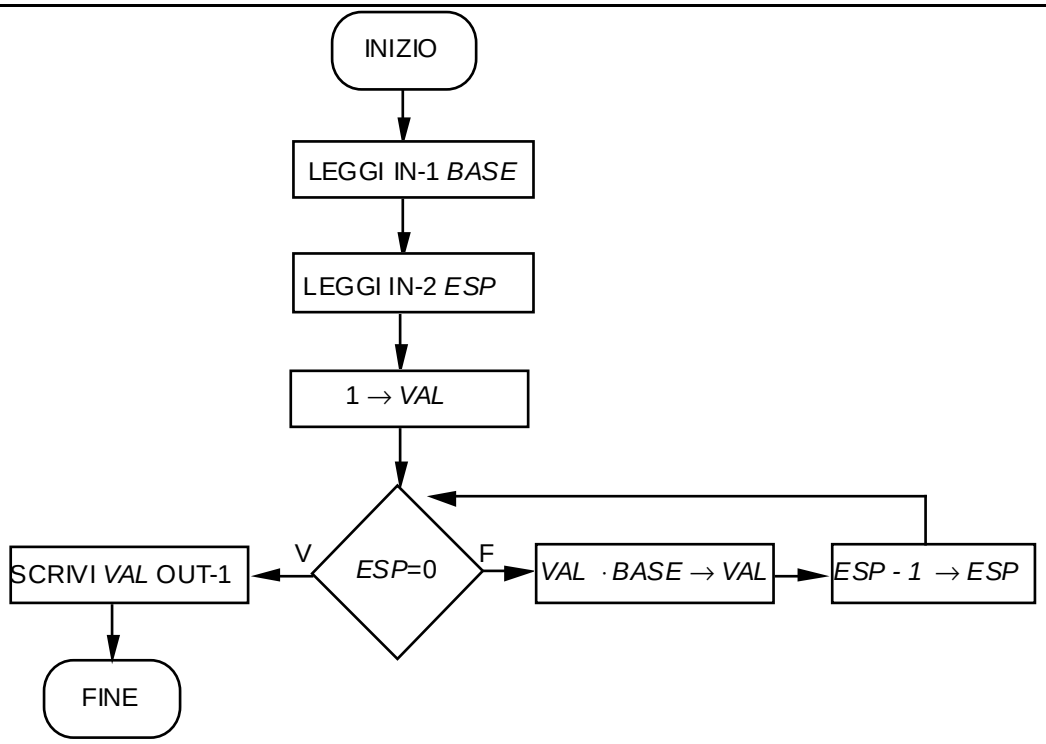

Figura 2.4 - Diagramma di flusso per il calcolo dell'n-esima potenza.

Dopo aver disegnato il diagramma di flusso sulla lavagna di lavoro dell'ufficio, il direttore avverte l'utente che l'ufficio è pronto a rispondere al suo quesito, e gli spiega le modalità dell'interazione con l'impiegato: dovrà recarsi allo sportello, scrivere la base della potenza richiesta nella prima riga del modulo IN, e l'esponente nella seconda riga dello stesso modulo. Una volta scritti i due numeri, dovrà premere il pulsante del segnalatore acustico, avvisando in questo modo l'impiegato che può iniziare a lavorare. Al termine del lavoro dell'ufficio, potrà leggere il risultato sul certificato OUT.

Supponiamo quindi che l'utente scriva i dati che gli interessano sul modulo IN, e cioè "4" nella prima riga e "3" nella seconda, dopodiché suoni il segnalatore acustico: l'impiegato si alzerà e piazzerà la freccia magnetica di fianco al blocco di inizio sulla lavagna di programma. Subito dopo la situazione dell'ufficio sarà quella illustrata in figura 2.5.

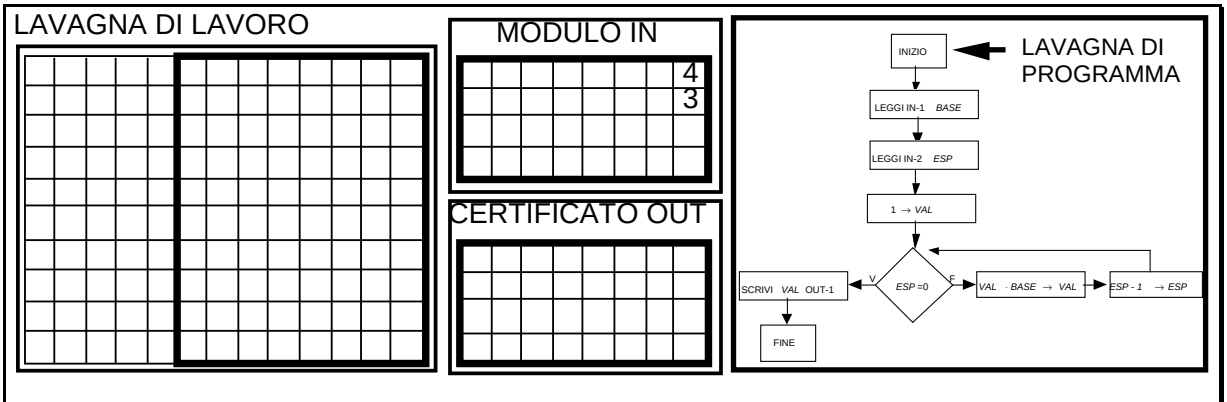

Figura 2.5

Il blocco trovato è un blocco funzionale l'istruzione elementare di lettura della base dalla prima posizione del modulo IN, e la sua scrittura nella posizione contrassegnata dal nome BASE sulla lavagna di lavoro. Poiché questo nome non è presente, viene scritto sulla cornice della lavagna nella prima posizione libera a partire dall'alto, ovvero in questo caso nella prima posizione, poiché non sono presenti altre variabili. Al termine dell'operazione, la situazione dell'ufficio è quella illustrata in figura 2.6. Terminata l'esecuzione del blocco funzionale, l'impiegato identifica il blocco successivo e posiziona la freccia magnetica al suo fianco.

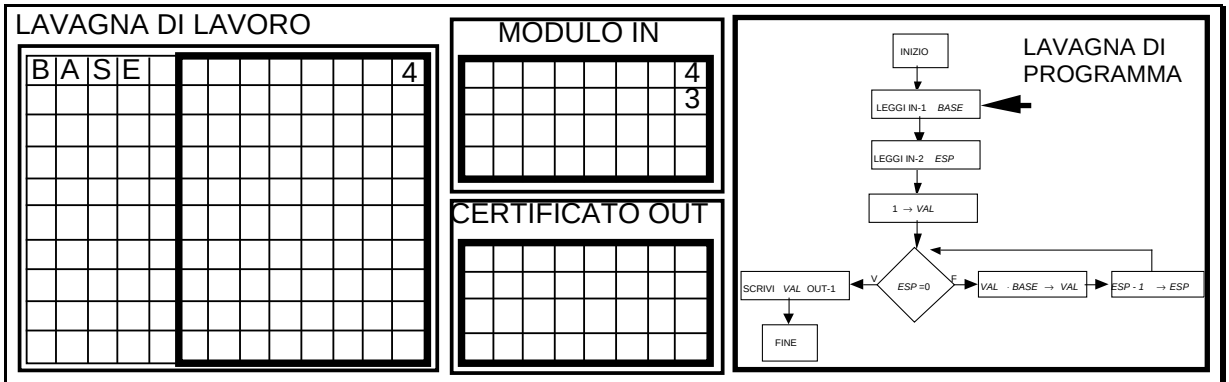

Figura 2.6

Il blocco funzionale successivo comporta la lettura dell'esponente dalla seconda posizione del modulo IN, e la sua scrittura sulla lavagna di lavoro nella posizione contrassegnata dal nome ESP; poiché anche questo non è presente, è necessario aggiungerlo sulla cornice della lavagna nella prima posizione libera, che questa volta è la seconda. La situazione dell'ufficio sarà quella di figura 2.7.

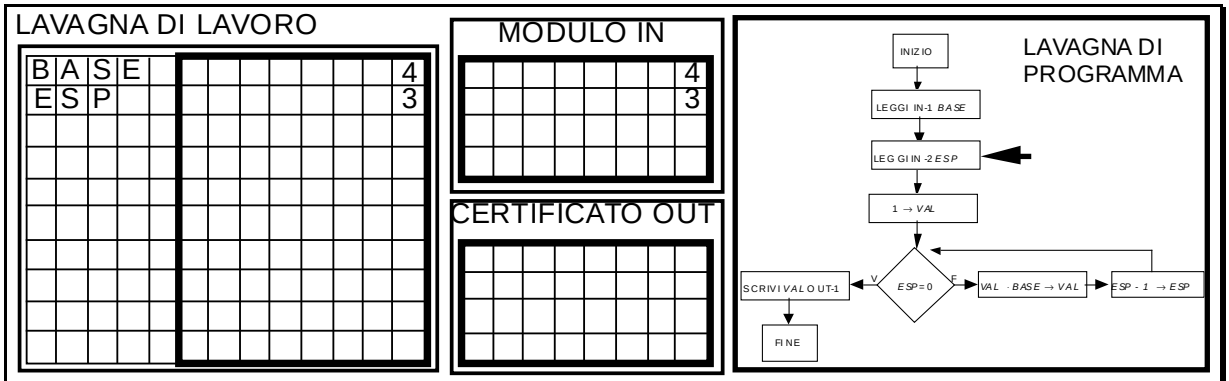

Figura 2.7

Il blocco che segue quello appena eseguito contiene un'istruzione che assegna alla variabile VAL il valore 1. Anche questa variabile non è mai stata allocata in precedenza, quindi sarà necessario allocarla (ovvero scriverne il nome sulla cornice alla prima posizione libera, la terza). L'esito è illustrato in figura 2.8.

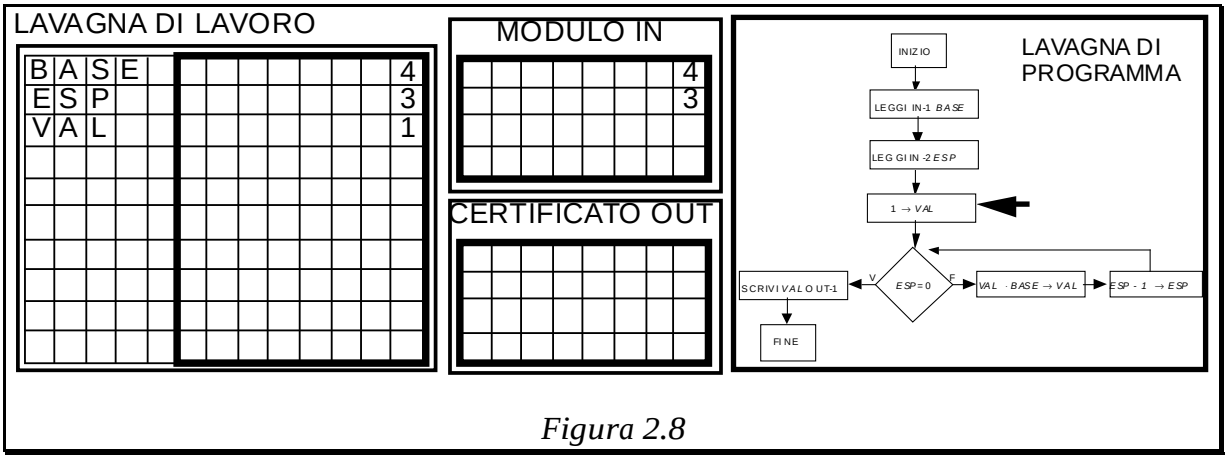

Figura 2.8

Terminata l'esecuzione del blocco funzionale, l'impiegato identifica il blocco successivo e posiziona la freccia magnetica al suo fianco (figura 2.9).

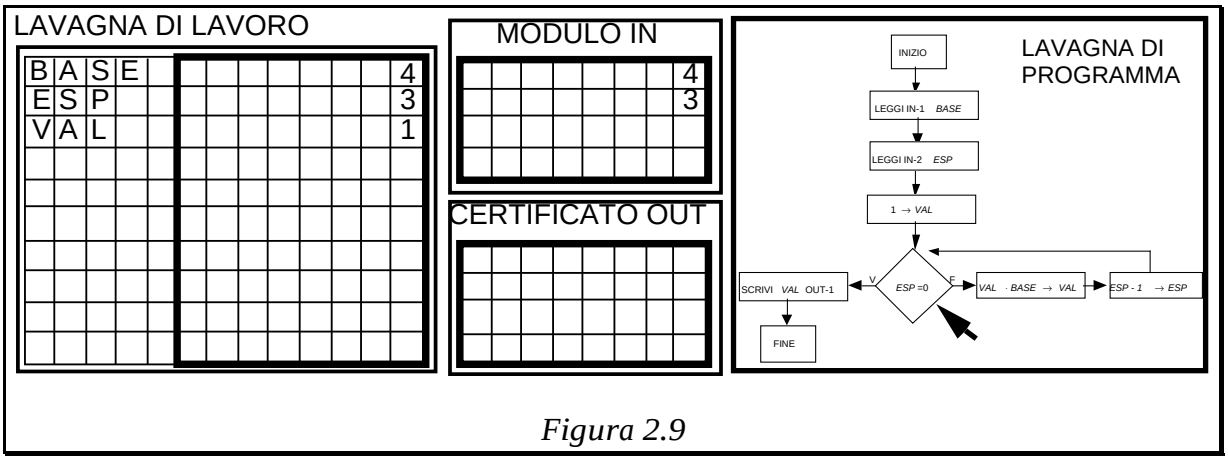

Figura 2.9

Il blocco in questione è di tipo decisionale, e contiene la condizione logica "ESP=0": l'impiegato legge il valore di ESP dalla lavagna di lavoro, cercandone il nome sulla cornice, e verifica se è effettivamente uguale a 0. Poiché il valore è 3, la condizione è falsa, e quindi identifica il blocco successivo seguendo la freccia contrassegnata con F (quella di destra), la quale porta a un nuovo blocco funzionale.

Il blocco funzionale successivo contiene un'istruzione di assegnamento: eseguendola, l'impiegato legge il contenuto di VAL e di BASE dalla lavagna di lavoro, li moltiplica l'uno per l'altro e scrive il risultato di nuovo in VAL, dopo avere cancellato il valore precedente. Il nuovo valore di VAL sarà quindi $1 \cdot 4 = 4$. Si noti che sia VAL sia BASE sono già presenti sulla cornice della lavagna, e quindi l'impiegato non ha bisogno di allocare nuove posizioni. Al termine del blocco il risultato è quello illustrato in figura 2.10.

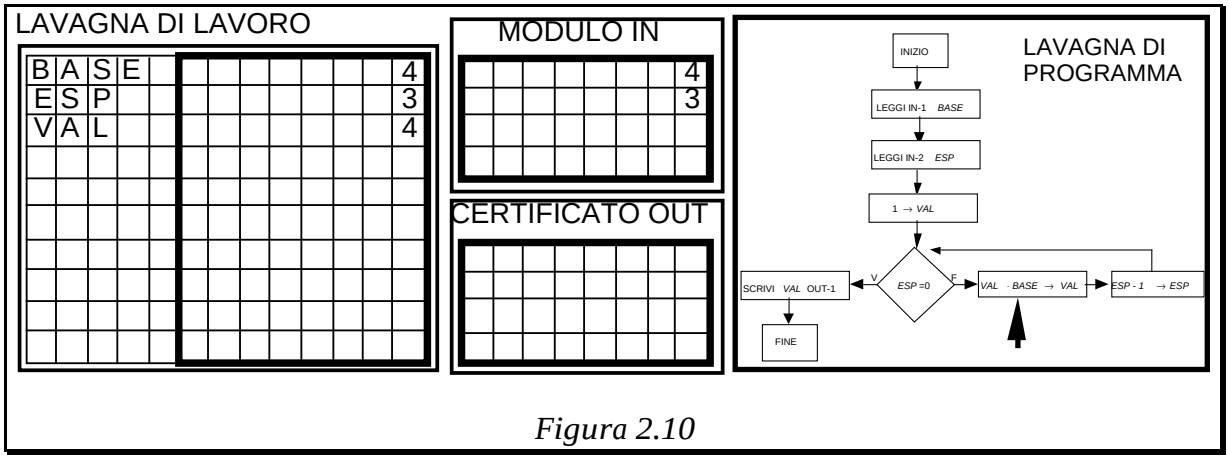

Figura 2.10

Il blocco seguente sarà di nuovo di tipo funzionale, e contiene un'ulteriore istruzione di assegnamento che decrementa di 1 il valore di ESP: l'impiegato legge il valore di ESP, gli sottrae 1 e riscrive il risultato (3 - 1 = 2) nella stessa posizione, dopo avere cancellato il valore precedente.

Lo stato dell'ufficio al termine di questo blocco funzionale è presentato in figura 2.11.

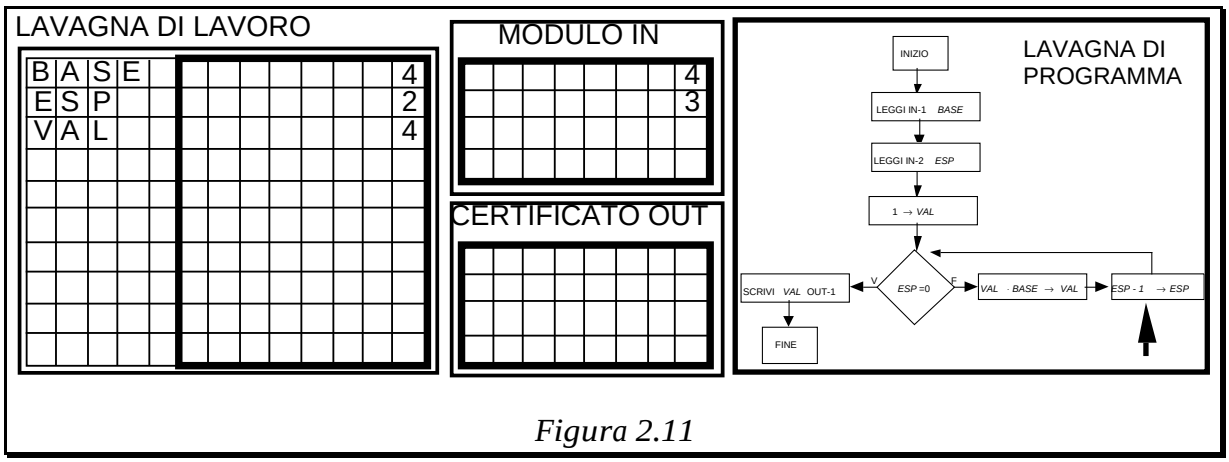

Figura 2.11

La freccia uscente dal blocco funzionale appena eseguito porta di nuovo al blocco decisionale incontrato in precedenza: quindi sarà necessario rivalutare la stessa condizione logica (figura 2.12). Il valore di ESP è cambiato rispetto alla situazione di figura 2.3, ed è pari a 2, ma la condizione logica rimane falsa, pertanto di nuovo l'impiegato dovrà posizionare la freccia magnetica sul blocco funzionale già incontrato.

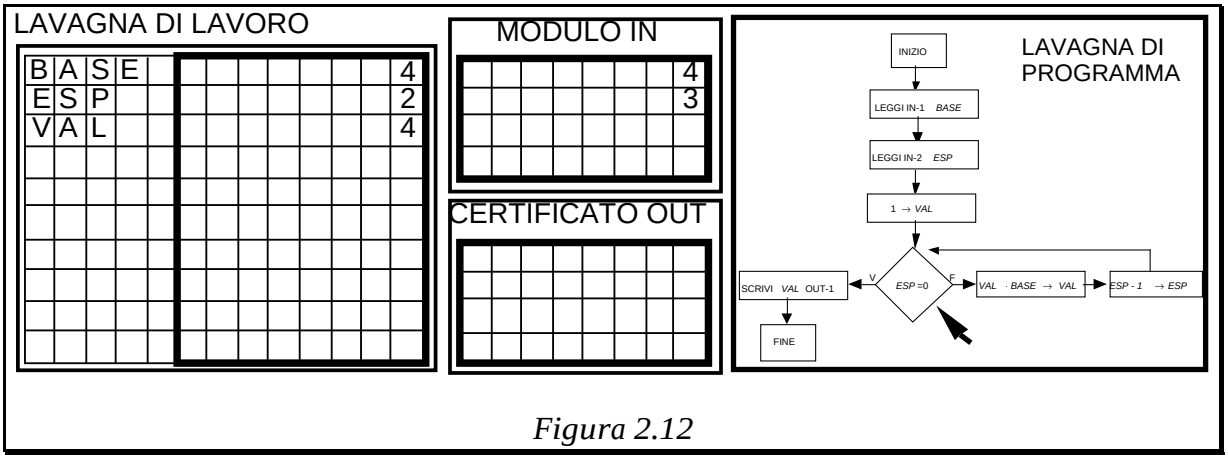

Figura 2.12

Di nuovo sarà quindi necessario valutare l'istruzione elementare di assegnamento contenuta nel blocco, con i valori diversi assunti dalle stesse variabili utilizzate in precedenza: quindi l'impiegato legge il contenuto di VAL e di BASE, li moltiplica l'uno per l'altro e scrive il risultato di nuovo in VAL, dopo avere cancellato il valore precedente. Il nuovo valore di VAL sarà quindi 4·4 = 16 (figura 2.13).

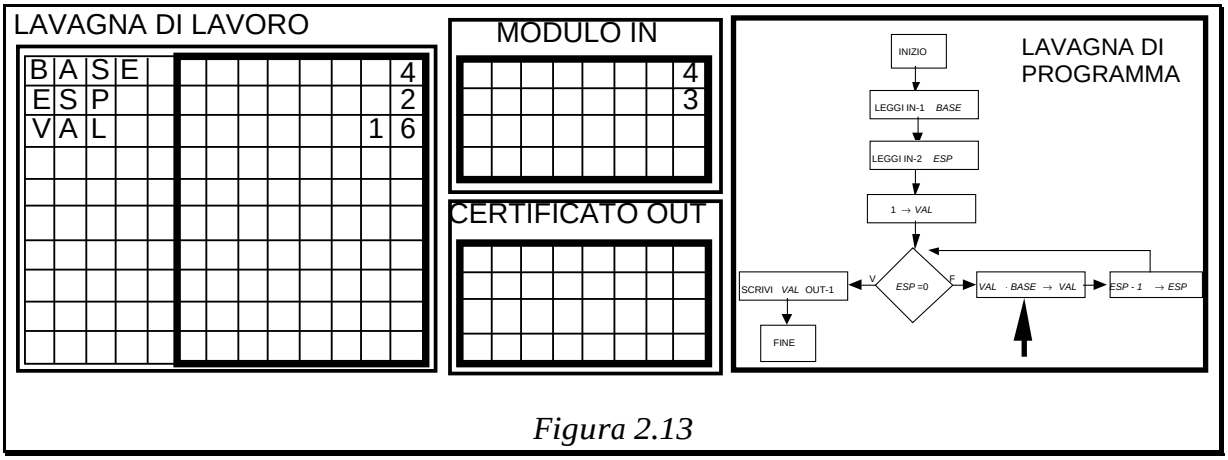

Figura 2.13

Posizionata la freccia magnetico sul blocco successivo, l'impiegato legge il valore di ESP, gli sottrae 1 e riscrive il risultato (2 - 1 = 1) nella stessa posizione, dopo avere cancellato il valore precedente.

L'esito di quest'ultima operazione dà luogo alla situazione dell'ufficio rappresentata in figura 2.14.

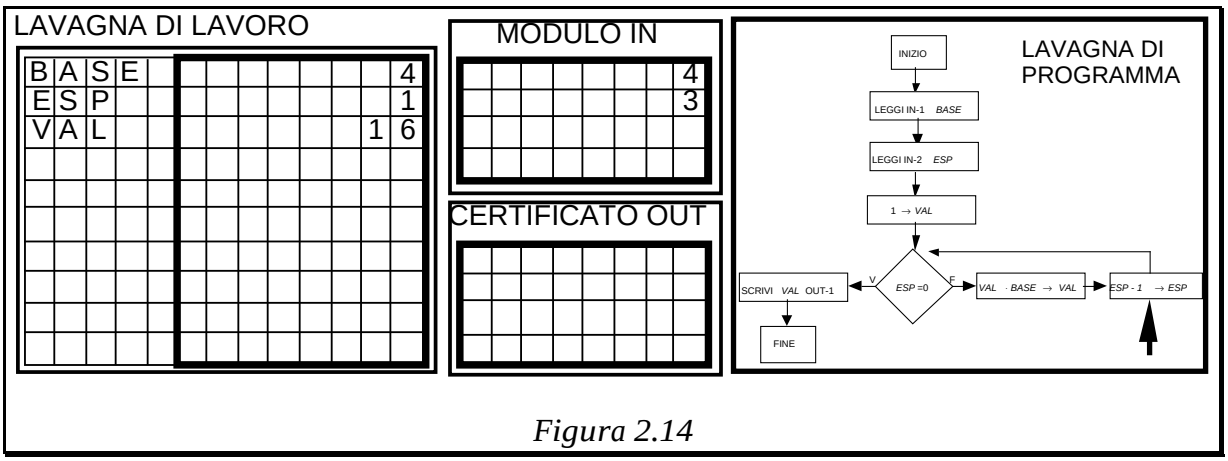

Figura 2.14

Di nuovo, il blocco collegato dalla freccia uscente al blocco funzionale appena eseguito è il blocco decisionale incontrato in precedenza. La condizione logica viene rivalutata con il nuovo valore assunto dalla variabile ESP (questa volta pari a 1), ed è ancora falsa (figura 2.15).

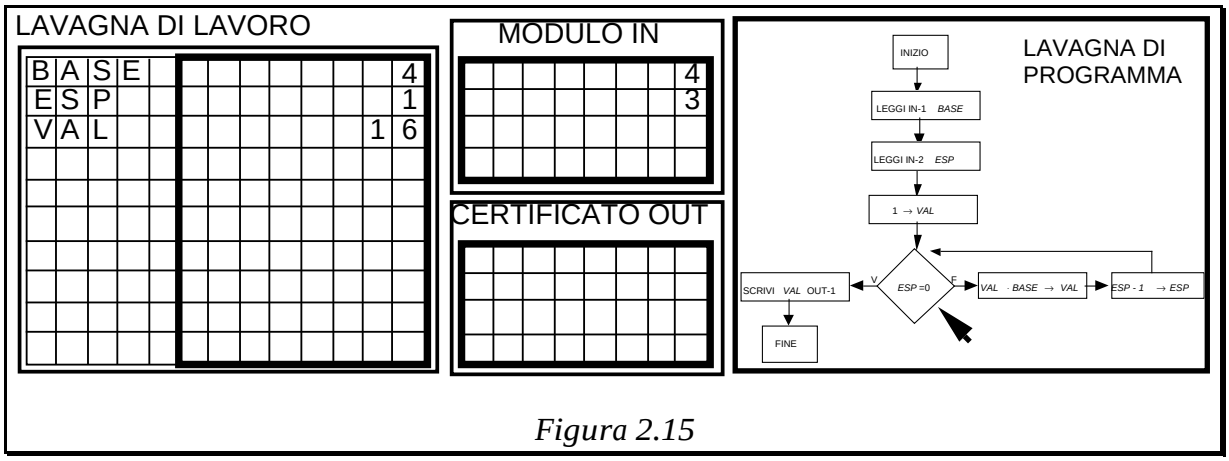

Figura 2.15

Di conseguenza, si tornerà a eseguire lo stesso blocco funzionale già eseguito in precedenza, che moltiplica il contenuto di VAL (=16) per il contenuto di BASE (=4), depositando il risultato di nuovo in VAL (16·4 = 64), dopo avere cancellato il valore precedente (figura 2.16).

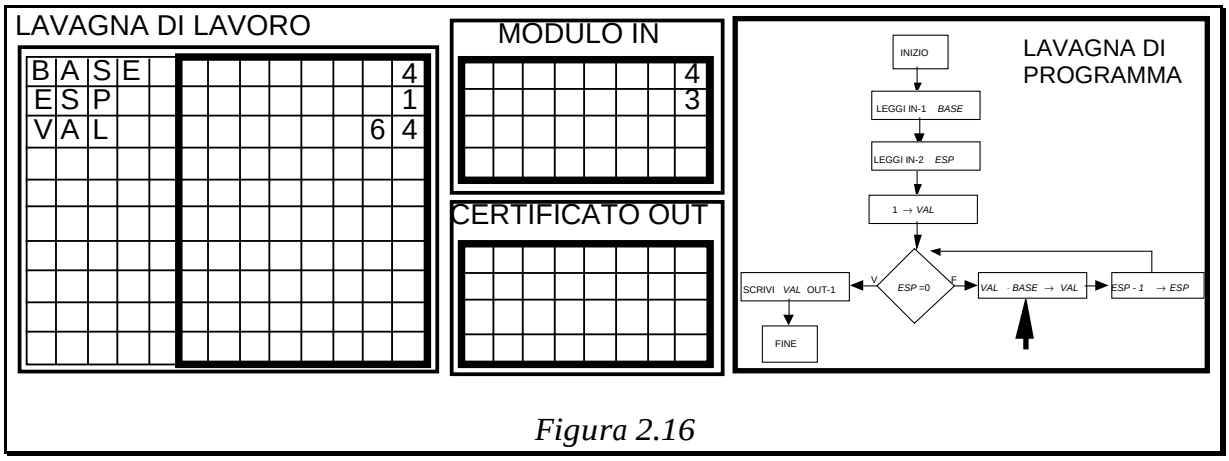

Figura 2.16

Il blocco successivo è sempre quello che decrementa il valore della posizione ESP: in questo caso, l'impiegato leggerà il valore di ESP (=1), gli sottrarrà il valore 1, e riscriverà il risultato (1-1=0) nella posizione ESP dopo avere cancellato il valore precedente. L'esito è in figura 2.17.

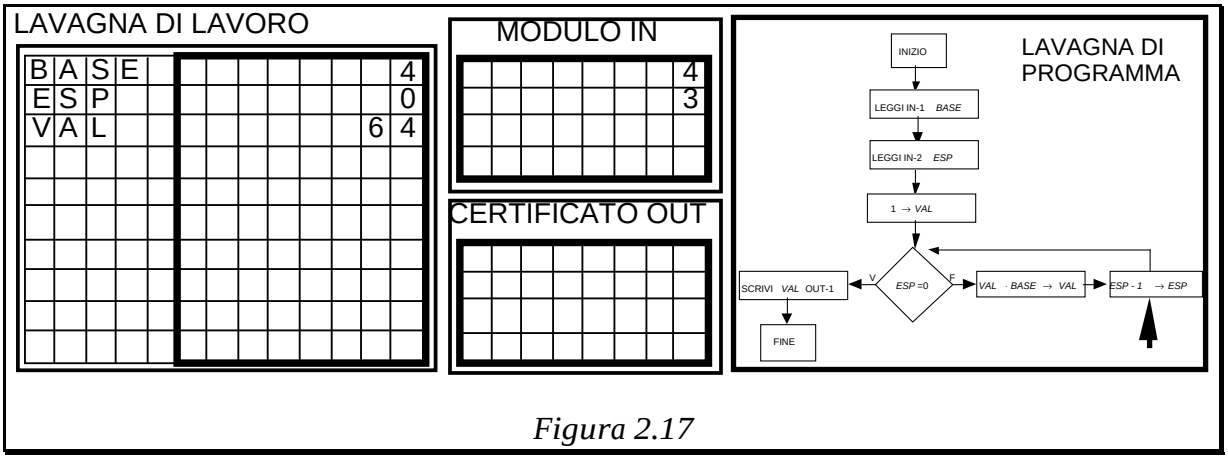

Figura 2.17

Di nuovo l'impiegato torna a valutare il blocco decisionale in cui si verifica se ESP è pari a 0 (figura 2.18).

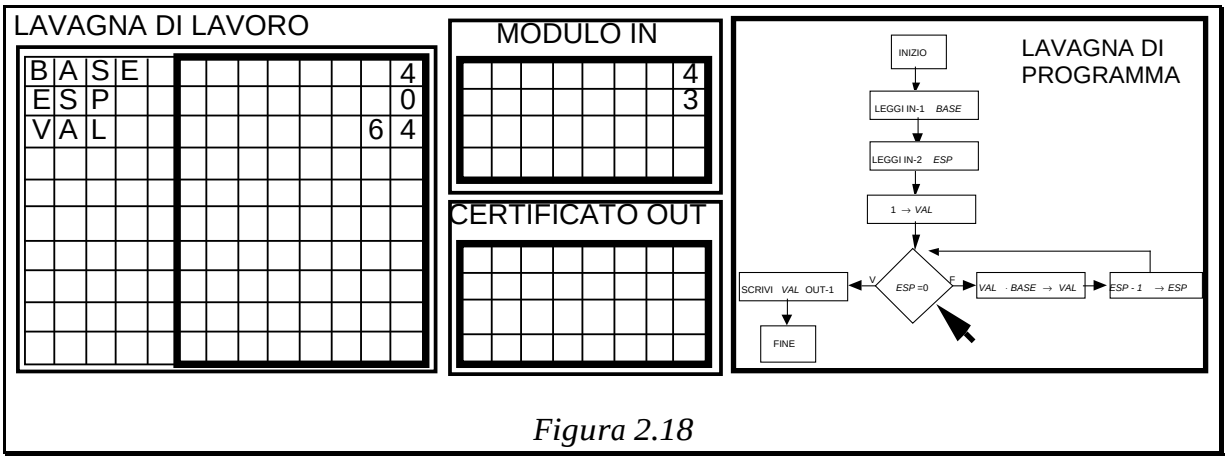

Figura 2.18

Con il valore corrente di ESP la condizione logica è vera, e quindi viene scelto l'arco uscente contrassegnato dall'etichetta "V", il quale porta a un nuovo blocco funzionale.

In questo blocco è presente un'istruzione di scrittura: l'impiegato quindi legge il valore contenuto sulla lavagna di lavoro alla posizione contrassegnata dal nome VAL, e lo scrive sul certificato OUT nella posizione OUT-1, cioé sulla prima riga. Il risultato di questa operazione è visibile in figura 2.19.

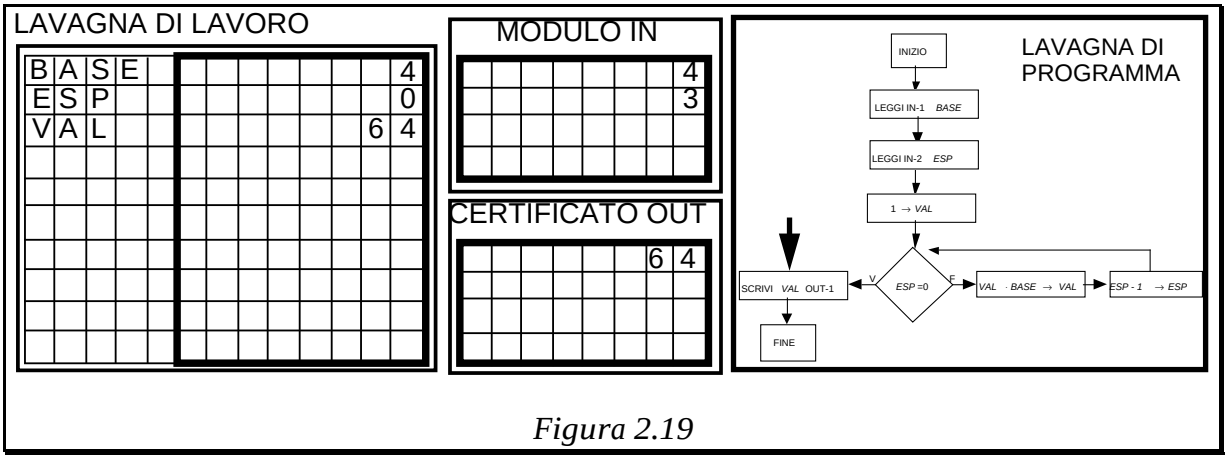

Figura 2.19

Il blocco collegato alla freccia uscente dal blocco appena eseguito è un blocco di fine, pertanto l'impiegato, dopo avere spostato la freccia magnetica, posiziona il certificato allo sportello e va a sedersi sulla sua sedia (fig. 2.20).

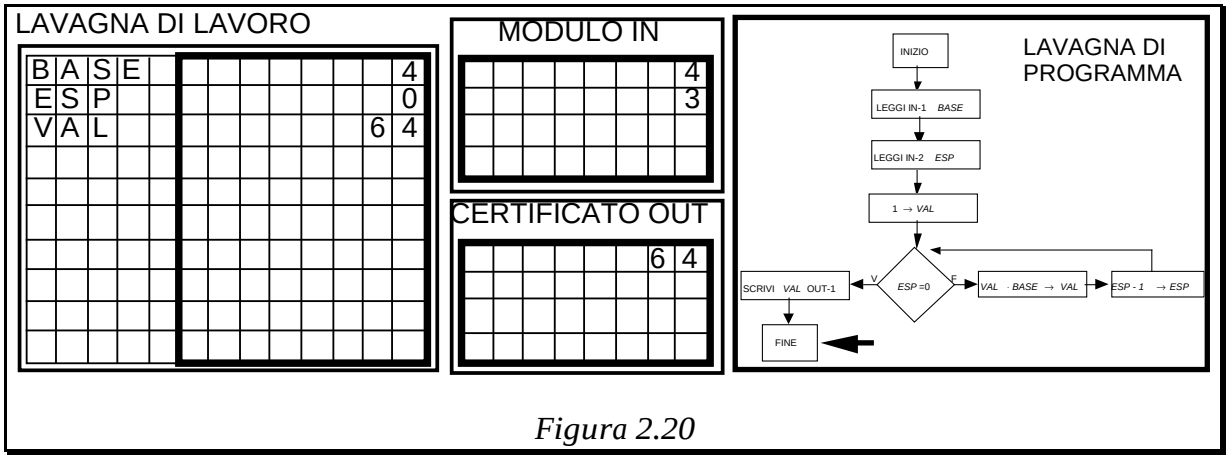

Figura 2.20

L'utente a questo punto avrà a disposizione un certificato sulla cui prima riga è scritto il risultato dell'elaborazione che aveva richiesto.

2.5 Caratteristiche delle attività dell'impiegato

Ora che abbiamo capito il modo di lavorare dell'impiegato, analizziamo più attentamente le caratteristiche dell'attività che egli svolge. L'impiegato:

- Conosce, capisce e sa eseguire solo le attività elementari elencate nel paragrafo 2.3. Esse non possono essere cambiate e l'impiegato non è in grado di inventarsi nuove istruzioni.
- Non ha capacità autonome (non prende nessuna decisione) e non può né cambiare il contenuto della lavagna di programma, né eseguire le istruzioni diversamente da come specificato.
- Non è in grado di gestire situazioni non previste o non coerenti con le sue regole di comportamento. Quando si verifica una situazione imprevista, egli si ferma (per esempio, lettura dalla lavagna IN da una posizione in cui non c'è alcun valore; fine delle posizioni disponibili sulla lavagna durante l'allocazione di una nuova posizione; risultato di un'operazione troppo grande per essere ospitato nelle caselle; divisione per 0; diagramma di flusso senza il blocco di inizio; ecc.).
- Sa comunque interpretare ed eseguire diagrammi di flusso di dimensioni arbitrarie senza commettere errori.

Possiamo dire che l'impiegato è, da un lato, efficiente in ciò che fa ma anche, dall'altro lato, molto rigido nel rispetto delle sue mansioni: possiamo aspettarci che esegua correttamente le procedure necessarie per svolgere i compiti di sua competenza, però non possiamo chiedergli niente che non sia specificato tra i suoi compiti.

Si osservi che non è necessario che l'impiegato abbia consapevolezza del proprio lavoro: egli conosce e sa eseguire le istruzioni elementari elencate nel paragrafo 2.3, ma è in grado di svolgere il suo compito anche senza capire il significato globale del diagramma di flusso. Ad esempio, egli può effettuare l'elevamento a potenza anche senza sapere che cosa questo sia: in un certo senso,

l'impiegato "non sa quello che fa". In realtà è più corretto dire che è l'ufficio a effettuare l'elevamento a potenza: ritorneremo nel capitolo 7 su questo punto.

Possiamo schematizzare l'attività dell'impiegato nel modo illustrato in figura 2.21.

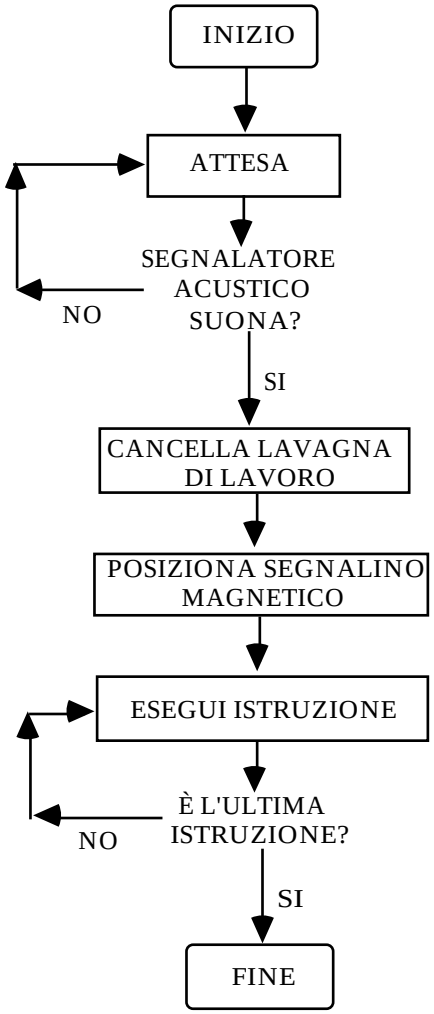

Fig. 2.21 - L'attività dell'impiegato.

2.6 Esercizi

Come vedremo nei prossimi capitoli, l'attività di un programmatore informatico è molto simile a quella del direttore. È quindi importante imparare a progettare diagrammi di flusso.

Esercizio 1. Si impersoni il direttore e si progetti il diagramma di flusso che calcola il massimo fra 10 numeri scritti nelle prime 10 posizioni del modulo IN.

Esercizio 2. Come fare per essere sicuri che il programma dell'esercizio precedente è corretto? Si impersoni l'impiegato e lo si esegua!

Esercizio 3. Si impersoni il direttore e si progetti il diagramma di flusso che calcola il massimo fra un numero arbitrario N di numeri, scritti nelle prime N posizioni del modulo IN (supponendo $N < M_{IN}$). Si controlli la correttezza

eseguendolo. (Suggerimento: bisogna che sul modulo di IN ci sia anche il valore di N).

Esercizio 4. Si risolva l'esercizio precedente per un valore completamente arbitrario di N, anche maggiore di M_{IN} (suggerimento: bisogna cambiare il metodo di interazione fra utente e ufficio).

Esercizio 5. Qual è il risultato restituito dall'impiegato se sulla lavagna di programma c'è il diagramma di figura 2.21 e sul modulo IN, nella posizione 1, c'è il numero 37?

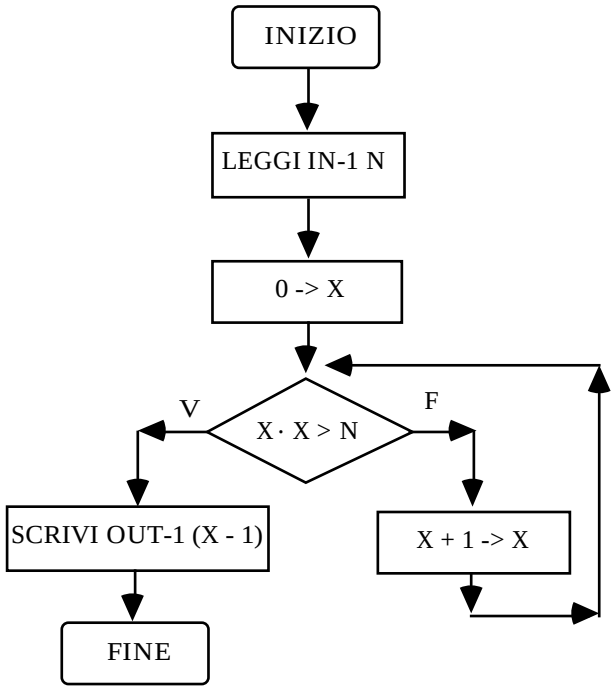

Fig. 2.22 - Diagramma di flusso per l'esercizio 5.

Esercizio 6. Qual è la funzione matematica calcolata dal diagramma di flusso dell'esercizio precedente?

Esercizio 7. Si progetti un diagramma di flusso per il calcolo della somma di N numeri.

Esercizio 8. Si progetti un diagramma di flusso per il calcolo della media aritmetica di N numeri.

Esercizio 9. Si scriva un diagramma di flusso per il calcolo del massimo comun divisore di due numeri il massimo comun divisore di due numeri naturali X e Y è il più grande numero Z che è un divisore sia di X sia di Y (N è un divisore di M se il resto di M / N è zero).

Esercizio 10. Si analizzi il diagramma di flusso precedente e si rifletta su come è possibile migliorarlo. Poi si progetti un diagramma di flusso per il calcolo del massimo comun divisore basato sulla proprietà seguente, dovuta al matematico greco Euclide:

- Se X = Y allora mcd(X,Y) = X = Y
- Se X > Y allora mcd(X,Y) = mcd(X - Y, Y)
- Se X < Y allora mcd(X,Y) = mcd(X, Y - X)

(la terza proprietà è la simmetrica della seconda). Si confrontino i vari diagrammi di flusso per calcolare il massimo comun divisore.

Esercizio 11. Si progetti un diagramma di flusso per verificare se un numero N è primo (ossia i suoi unici divisori sono 1 e N).

Esercizio 12. Si progetti un diagramma di flusso che dati un numero X e una sequenza di N numeri N_1, ..., N_N, ricerchi X fra gli N numeri e ne stampi sul modulo di OUT la posizione.

Esercizio 13 (*). Come si può migliorare il diagramma di flusso dell'esercizio precedente se gli N numeri sono ordinati (ad esempio, in modo crescente)? (Suggerimento: si confronta X con il numero mediano $N_{N/2}$; se X = $N_{N/2}$, siamo stati fortunati e la posizione è appunto N/2; se X < $N_{N/2}$, si può scartare la seconda metà dei dati, che sono tutti maggiori o uguali di $N_{N/2}$; se X > $N_{N/2}$ si scarta la prima. Si ripete poi il procedimento sulla metà non scartata, confrontando X con l'elemento a metà della metà, e così via. Il diagramma di flusso dell'esercizio precedente realizza la cosiddetta ricerca *lineare*; quello di questo esercizio la ricerca *binaria*.)

Esercizio 14. Qual è il risultato del diagramma di flusso di figura 2.23 se sulla prima posizione del modulo di IN c'è il valore 3? E se c'è il valore 34? Si rifletta su questa seconda possibilità...

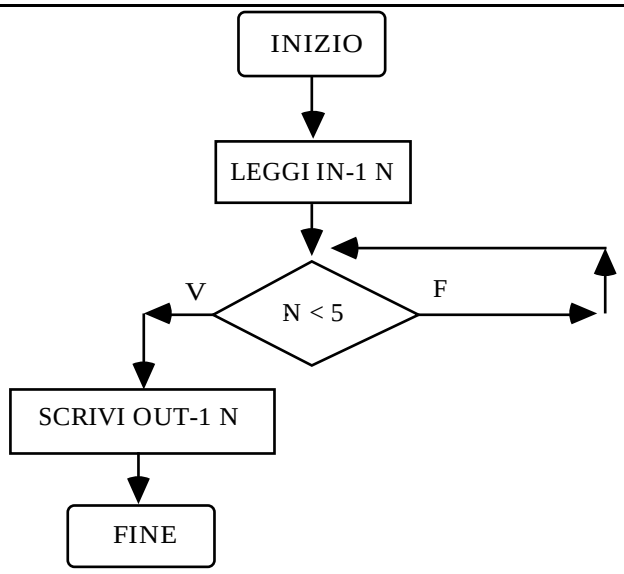

Figura 2.23 - Un diagramma di flusso apparentemente innocuo...

Esercizio 15. Quanti sono i passi compiuti dall'impiegato durante l'esecuzione del diagramma di flusso del paragrafo 2.4? (Risposta: 7 + (IN-2 * 3) istruzioni).

Capitolo 3
Il concetto di algoritmo

In questo capitolo possiamo finalmente illustrare il concetto di algoritmo, utilizzando la metafora dell'ufficio.

3.1 L'attività dell'ufficio e il ruolo del direttore

Abbiamo osservato nel paragrafo 2.5 come l'impiegato non sappia fare altro che svolgere i suoi compiti elementari (le poche operazioni aritmetiche e di ingresso/uscita illustrate nel paragrafo 2.3) e come l'ufficio nel suo complesso sia invece in grado di effettuare operazioni più complesse, ad esempio l'elevamento a potenza o le varie funzioni matematiche illustrate negli esercizi del paragrafo 2.6. Ciò è possibile perché l'ufficio è in un certo senso "istruito" ("programmato" è il termine corretto, come vedremo) dal direttore: il direttore deve progettare una sequenza opportuna di operazioni elementari, combinate opportunamente, e illustrare questa sequenza sulla lavagna di programma per mezzo del formalismo dei diagrammi di flusso.

Il complesso ufficio-impiegato-direttore è quindi in grado di risolvere alcuni problemi o, detto in altro modo, di fornire la risposta ad alcune domande. Ma l'approccio seguito è particolare: il direttore non progetta un meccanismo per rispondere a una specifica domanda, bensì cerca di istruire l'ufficio con un diagramma di flusso che sia in grado di rispondere a un gruppo di domande simili. Si riconsideri l'esempio dell'elevamento a potenza del paragrafo 2.4: il diagramma di flusso costruito dal direttore è in grado di calcolare il valore di M^N per qualsiasi M e N positivi, non solo per uno specifico valore di M e N. Così facendo, si ottiene un procedimento che può essere usato più volte per rispondere a domande dello stesso gruppo; ovviamente per rispondere a domande completamente differenti bisognerà progettare un nuovo diagramma di flusso.

Intuitivamente, è chiaro che è ben diverso dare risposte a domande specifiche ("Quanto fa 3^4?") o progettare un diagramma di flusso in grado di fornire la risposta a una qualsiasi domanda del tipo "Quanto fa M^N?". Introduciamo una terminologia per descrivere in modo più preciso queste problematiche.

3.2 Problemi, domande e risposte

Un *problema* è un'insieme di domande omogenee. Ad esempio "Calcolare la radice quadrata intera di un numero naturale" (chiamiamo questo problema P1,

per comodità) o “Calcolare il massimo comun divisore di due numeri naturali” (problema P2), ecc. Alle varie domande è possibile dare una risposta mediante un procedimento comune.

Un’*istanza* di un problema è uno specifico esemplare della classe di domande omogenee, una specifica domanda. Ad esempio “Calcolare la radice quadrata intera di 27” (istanza I1) o “Calcolare il massimo comun divisore di 21 e 63” (istanza I2) sono rispettivamente istanze dei problemi P1 e P2 appena visti. Oltre all’istanza I1, il problema P1 ammette altre infinite istanze (una per ogni numero naturale). Rispettando la terminologia adottata comunemente, useremo spesso il termine *domanda* per riferirci a un’istanza. La figura 3.1 illustra in modo schematico problemi e domande.

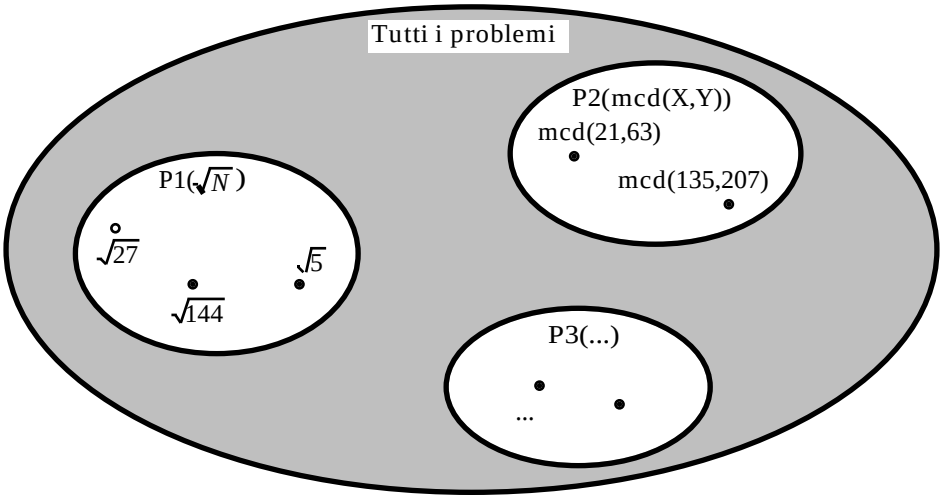

Fig. 3.1 - Problemi e domande.

Nella descrizione di un problema vi sono solitamente certi punti che vengono lasciati non precisati. Ad esempio, in “Calcolare la radice quadrata intera di un numero naturale” non compare il valore del numero naturale. Per descrivere un problema in modo più preciso si possono evidenziare questi punti usando il concetto di *variabile*: i due problemi precedenti potrebbero essere formulati con “Calcolare la radice quadrata intera di un numero naturale X” (che potrebbe essere etichettato con P1[X]) e “Calcolare il massimo comun divisore di due numeri naturali X e Y” (P2[X,Y] per evidenziare che vi sono due variabili). Queste variabili sono dette *variabili d’ingresso* e i valori che esse assumono su una specifica istanza *dati*. La loro utilità si manifesta chiaramente se si prova a esprimere il problema “Dati 6 numeri interi N1, N2, N3, N4, N5 e N6, calcolare l’espressione

$$(4 \cdot N1 + (N2^{N3})) \,/\, (N4 + N5 \cdot N6 + N1)$$”

senza farne uso.

Inoltre, evidenziando esplicitamente i dati di un problema, si comprende meglio il legame fra problema e istanza: un’istanza deve specificare il valore di queste variabili. Potremmo scrivere I1 = P1[27], evidenziando che I1 è l’istanza di P1 con X = 27, e similmente I2 = P2[21,63]. Se non indichiamo il valore di *tutte* le variabili (ad esempio, P2[21,Y]) non otteniamo un’istanza ma ancora un problema, che possiamo chiamare *sottoproblema* del problema originario (P2[X,Y]). Il sottoproblema è ovviamente più semplice del problema.

La *soluzione di un'istanza* di un problema è la risposta a una domanda. Ad esempio, "5" e "21" sono le rispettive soluzioni delle istanze I1 e I2 precedenti. Diremo spesso *risposta* o *risultato* anziché soluzione di un'istanza.

3.3 Algoritmi, linguaggi di programmazione e programmi

La *soluzione di un problema* (o *algoritmo risolvente*) è un metodo generale che permette di fornire in modo uniforme la risposta a tutte le istanze di un problema. Si osservi che l'algoritmo è un'entità astratta: è un "metodo", un "procedimento", una "sequenza di passi". Non è possibile vedere o toccare un algoritmo, al massimo si può percepirne l'esecuzione (ad esempio osservando l'impiegato che effettua certi passi) o osservarne una rappresentazione tangibile (ad esempio in termini di diagrammi di flusso, ma non solo). Quindi l'algoritmo rappresentato dal diagramma di flusso dell'esercizio 5 del paragrafo 2.6 è la soluzione del problema P1.

Abbiamo quindi capito che un algoritmo è un procedimento che permette di rispondere a tutte le istanze di un certo problema. Essendo però l'algoritmo un concetto astratto, non può essere comunicato dal direttore all'ufficio senza un passo di *rappresentazione*. Ed è proprio ciò che viene fatto dal direttore quando scrive il diagramma di flusso sulla lavagna di programma. Il diagramma di flusso è quindi la rappresentazione dell'algoritmo in un certo formalismo. La rappresentazione è detta *programma* e il formalismo *linguaggio di programmazione*.

Notiamo per inciso che l'algoritmo illustrato nel paragrafo 2.4 per calcolare la potenza di due numeri naturali può essere rappresentato anche in altri modi. Ad esempio, potremmo descriverlo utilizzando il linguaggio di tutti i giorni, magari nel modo seguente:

- leggi i due numeri;
- se il secondo numero è zero, stampa 1;
- altrimenti moltiplica il primo numero per sè stesso tante volte quanto indicato dal secondo numero e stampa il risultato ottenuto.

Anche questa è una rappresentazione di quello che potremmo chiamare "l'algoritmo per l'elevamento a potenza per moltiplicazioni ripetute". Ma essa è piuttosto diversa da quella creata dal direttore e rappresentata in figura 2.4:

- è a un più alto livello di astrazione, più vicina al nostro modo di parlare e ragionare, e quindi probabilmente più comprensibile per noi umani;
- è però meno precisa, più ambigua;
- non è espressa in termini di istruzioni elementari che l'impiegato sa eseguire.

Ritorneremo fra poco su questi argomenti. Si noti comunque che possono esistere più linguaggi di programmazione e che uno stesso algoritmo può essere rappresentato in più linguaggi di programmazione, ottenendo quindi programmi diversi. Inoltre uno stesso algoritmo può essere rappresentato con due programmi diversi scritti nello stesso linguaggio di programmazione: un modo banale per convincersi di ciò è aggiungere istruzioni che non modificano il risultato (ad esempio, incrementano di 1 il valore di una variabile non utilizzata in altri punti del programma). La figura 3.2 schematizza quanto detto finora.

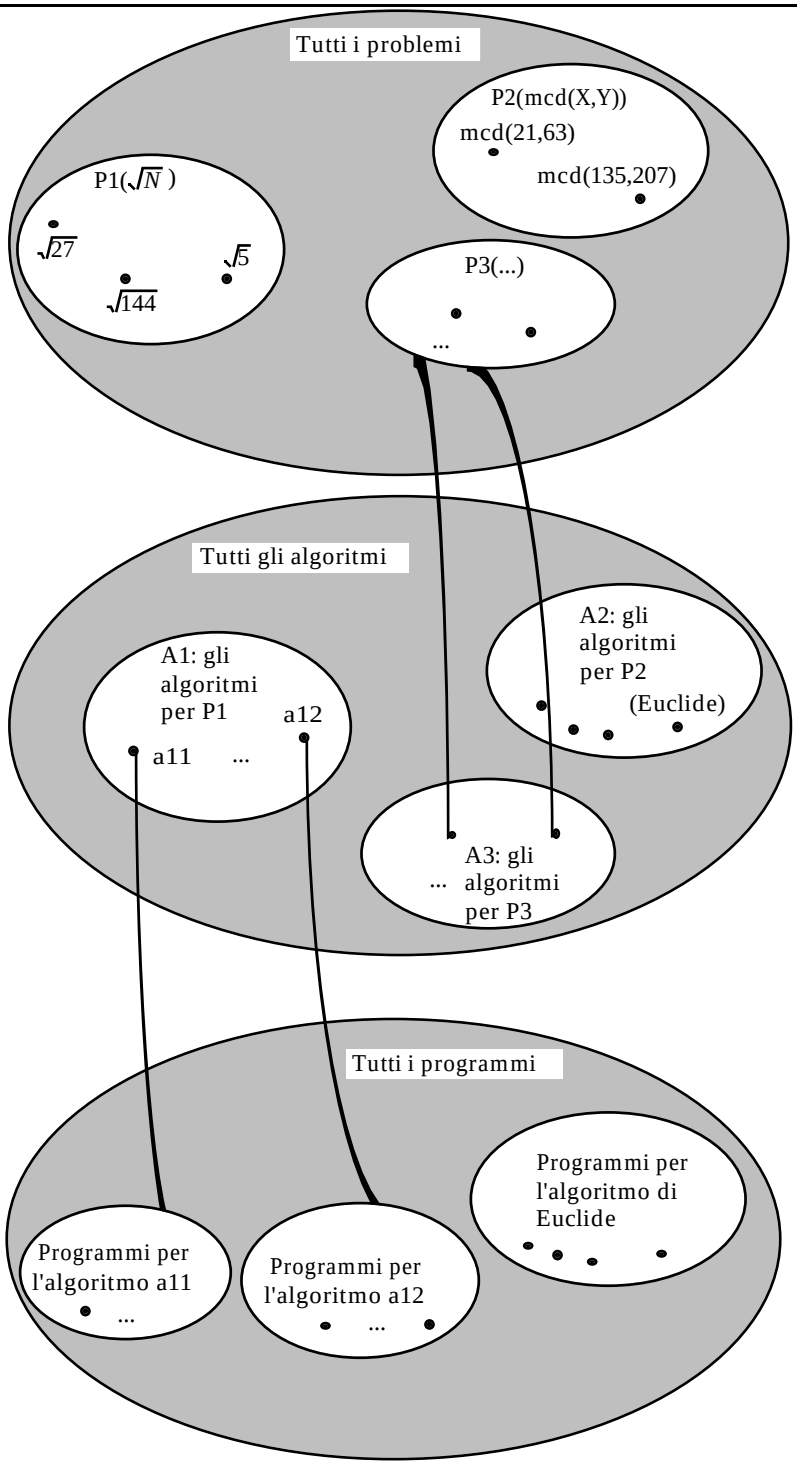

Fig. 3.2 - Problemi, domande, algoritmi, programmi.

3.4 Calcolatori

Ma che cos'è un *calcolatore*? A questo punto possiamo rispondere: è un esecutore di algoritmi rappresentati mediante un linguaggio di programmazione. Più precisamente, è un sistema che:

- riceve un programma e dei dati; il programma è la rappresentazione di un algoritmo (in un linguaggio di programmazione comprensibile dal

calcolatore) che risolve un dato problema, e i dati sono i valori che identificano una specifica istanza;
- esegue il programma sui dati;
- fornisce quindi la risposta all'istanza del problema derivata istanziando il problema sui dati.

Un calcolatore è quindi un *esecutore universale* di algoritmi: può eseguire qualsiasi algoritmo gli venga fornito, purché rappresentato nel formalismo che il calcolatore sa comprendere. L'attività di un calcolatore è illustrata in figura 3.3.

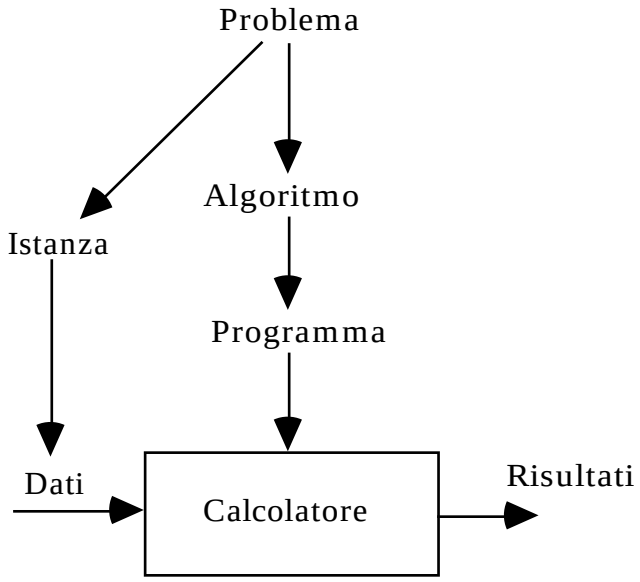

Fig. 3.3 - L'attività del calcolatore.

L'esecuzione di un programma su un certo calcolatore/esecutore è detta *computazione*. Si noti quindi che algoritmo e programma sono concetti *statici*, mentre la computazione è un concetto dinamico.

Il nostro ufficio è quindi un calcolatore; il linguaggio di programmazione che egli sa comprendere è quello dei diagrammi di flusso. Il direttore è il *programmatore* di questo calcolatore, ossia la persona che deve rappresentare l'algoritmo che risolve il problema dell'utente in modo comprensibile dal calcolatore.

3.5 Proprietà formali degli algoritmi

Un algoritmo deve quindi essere un metodo, effettivamente eseguibile da un calcolatore, per risolvere un problema. È possibile elencare alcune proprietà, caratteristiche, che un algoritmo (o forse sarebbe meglio dire programma? ritorneremo a fine capitolo su questo punto) *deve* avere. Un algoritmo deve essere:

- **Finito.** Un algoritmo deve essere composto da un numero finito di istruzioni.

- **Non ambiguo.** Ogni istruzione deve essere non ambigua, interpretabile in un unico modo. Inoltre anche il modo di combinare le istruzioni deve essere non ambiguo.
- **Effettivo.** Deve esistere un calcolatore in grado di eseguire effettivamente l'algoritmo.

Discutiamo brevemente ciascuna di queste proprietà.

3.5.1 Finitezza

Se un algoritmo potesse essere composto da un numero infinito di istruzioni, non vi sarebbe un calcolatore in grado di eseguirlo. Qualsiasi calcolatore reale può infatti eseguire, in un intervallo di tempo finito, solo un numero finito di istruzioni. Ma non solo: come si potrebbe rappresentare un algoritmo infinito in un linguaggio di programmazione? Solo con un numero infinito di istruzioni! Il che vorrebbe dire che il direttore del nostro ufficio dovrebbe impiegare un tempo infinito per rappresentare l'algoritmo sulla lavagna di programma. E inoltre: come potrebbe l'impiegato trovare il blocco di inizio sulla lavagna di programma se questa contiene un diagramma con un numero infinito di blocchi? L'impiegato avrebbe bisogno di un tempo infinito per farlo!

Un esempio di "algoritmo" infinito (che in realtà, appunto perché infinito, non è un algoritmo) potrebbe essere il seguente, per risolvere il semplice problema del calcolo del successore di un numero N:

Se N = 0 allora stampo 1
Se N = 1 allora stampo 2
Se N = 2 allora stampo 3
Se N = 3 allora stampo 4
... e così via ...

Ma un tale "algoritmo" sarebbe completamente inutilizzabile da un calcolatore.

La finitezza è anche la ragione per cui il nostro ufficio lavora su numeri interi ("senza la virgola"). Infatti, alcuni numeri con la virgola hanno una rappresentazione infinita (si pensi al π, o al numero e, o alla $\sqrt{2}$): come potrebbe il nostro povero impiegato scrivere, o leggere, in un tempo finito un numero infinito di cifre?

3.5.2 Univocità

Se un algoritmo (o meglio un programma) contenesse un'istruzione ambigua, o più istruzioni combinate in modo ambiguo, il nostro impiegato non saprebbe come comportarsi, e si bloccherebbe. Un tale "algoritmo" non produrrebbe quindi risultati.

Un esempio di istruzione ambigua potrebbe essere una condizione del tipo *X è grande abbastanza*?, oppure un'operazione come X + *"un po'"* $\rightarrow X$. Un esempio di istruzioni combinate in modo ambiguo potrebbe essere *Ripeti per un po' di volte le istruzioni seguenti* (quante volte? quali istruzioni?).

3.5.3 Effettività

È necessario che esista un calcolatore in grado di eseguire effettivamente ogni istruzione dell'algoritmo finito in un tempo finito, in modo che si abbia la garanzia che l'algoritmo sia eseguito in un tempo finito. Ciò si ottiene richiedendo quattro sottoproprietà:

- La complessità delle istruzioni elementari eseguibili dall'esecutore (nel nostro caso, l'impiegato) deve essere limitata. Ossia, non è possibile immaginarsi un impiegato in grado di eseguire istruzioni arbitrariamente complesse. Infatti, se ciò fosse possibile, ogni problema sarebbe risolto da un'unica istruzione (ad esempio, *Calcola la radice quadrata*) e il concetto di algoritmo perderebbe significato. Inoltre il tempo necessario per eseguire un'istruzione arbitrariamente complessa sarebbe potenzialmente infinito.
- Deve esistere un limite finito ai tipi di istruzioni eseguibili; altrimenti l'esecutore impiegherebbe un tempo infinito a selezionare l'istruzione da eseguire.
- L'esecutore deve disporre di una memoria illimitata, cioé finita a ogni istante, ma per la quale non esiste limite.
- Inoltre, un'altra limitazione che si impone all'esecutore di algoritmi è l'operare per passi *discreti*, un passo dopo l'altro.

3.6 Una definizione formale di algoritmo?

Il lettore attento avrà senz'altro notato che finora non abbiamo definito in modo formale, preciso il concetto di algoritmo. In realtà, siccome questo concetto è troppo astratto, quello che si riesce a fare è dare definizioni formali di "programma", non di algoritmo. Si potrebbe definire un algoritmo come l'insieme di tutti i programmi "isomorfi"... **[DA COMPLETARE...]**

3.7 Esercizi

[DA COMPLETARE...]

Capitolo 4
Cenni di teoria della calcolabilità

Il lettore curioso si potrebbe senz'altro chiedere: "Ma con gli algoritmi si può fare tutto?". E, se la risposta è no, "Che cosa si può fare?" O anche, dato che (come vedremo nel prossimo capitolo) esistono più formalismi differenti per rappresentare algoritmi: "Esistono algoritmi che non sono rappresentabili nel formalismo che usiamo noi ma lo sono in un altro?". Queste sono domande legittime e molto interessanti e sono studiate dalla disciplina nota con il nome di *teoria della calcolabilità* (o *computabilità*). In questo capitolo cerchiamo di rispondervi in modo rigoroso, seppure sempre in modo intuitivo.

4.1 Gli algoritmi calcolano funzioni

Cerchiamo innanzi tutto di formulare le domande precedenti in modo più preciso, sfruttando un concetto matematico assodato, quello di *funzione*. Una funzione è una legge che associa a un elemento di un insieme detto *dominio* un altro elemento di un altro insieme detto *codominio* (si veda la figura 4.1). Ad esempio, potremmo avere la funzione $f_1(x)$ che associa a un numero x il suo successore; la funzione $f_2(x)$ che associa a un numero x la sua radice quadrata; e così via. Per rappresentare una funzione f con dominio D e codominio C, si scrive f: $D \rightarrow C$.

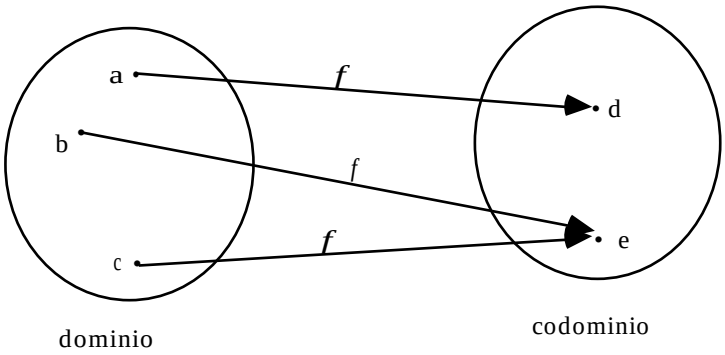

Figura 4.1 - Rappresentazione grafica di una funzione.

Una funzione può avere come dominio (o codominio) un insieme più complicato di quanto detto finora. Ad esempio, la funzione *potenza* associa a due numeri x e y il valore x^y, e quindi ha come dominio due insiemi. Per

rappresentare una funzione *f* con dominio costituito da due insiemi D_1 e D_2 si usa la simbologia $f:D_1 \times D_2 \rightarrow C$. La figura 4.2 rappresenta graficamente la funzione potenza $f:N \times N \rightarrow N$.

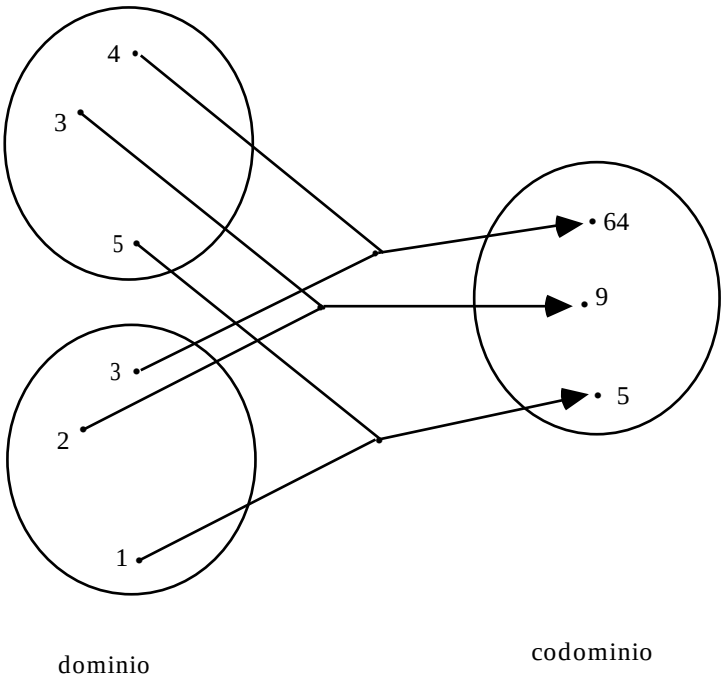

Figura 4.2 - Rappresentazione grafica della funzione potenza.

Finora abbiamo sempre visto gli algoritmi come "risolutori di problemi" o "esecutori di compiti". Ma un algoritmo può anche essere visto come un oggetto che calcola una *funzione*. Ad esempio, l'algoritmo per calcolare la radice quadrata calcola la funzione f_2; l'algoritmo del paragrafo 2.4 calcola la funzione $f_3(x,y)$ che associa a una coppia di numeri (x e y) il valore x^y; e così via.

Le funzioni di interesse della teoria della calcolabilità sono particolari: lavorano su numeri naturali (numeri "senza la virgola" e positivi).[2] Inoltre possono non essere definite su alcuni valori del dominio; in questo modo possono rappresentare la situazione di un programma che non dà nessun risultato in output (ad esempio perché esegue ripetutamente all'infinito la stessa istruzione). Funzioni con questa caratteristica (non definite su alcuni valori del dominio) sono dette *parziali*.

Inoltre, possiamo restringere la nostra analisi alle funzioni che associano numeri naturali a numeri naturali (indicate con $f: \boldsymbol{N} \rightarrow \boldsymbol{N}$). Infatti, data una funzione che associa a più numeri naturali un numero naturale, utilizzando tecniche di codifica, è sempre possibile associarvi in modo univoco una funzione $f: \boldsymbol{N} \rightarrow \boldsymbol{N}$ con proprietà analoghe. La figura 4.3 rappresenta l'insieme delle funzioni che ci interessa studiare.

[2]Questa potrebbe sembrare di primo acchito una limitazione forte, dato che i calcolatori odierni sono in grado di gestire anche numeri con la virgola, testi, suoni, immagini e video. Ma come illustrato in qualsiasi testo di informatica di base (ad esempio [Con97]) i calcolatori rappresentano al loro interno tutti i dati in forma digitale, e il considerare solo numeri interi non è una limitazione.

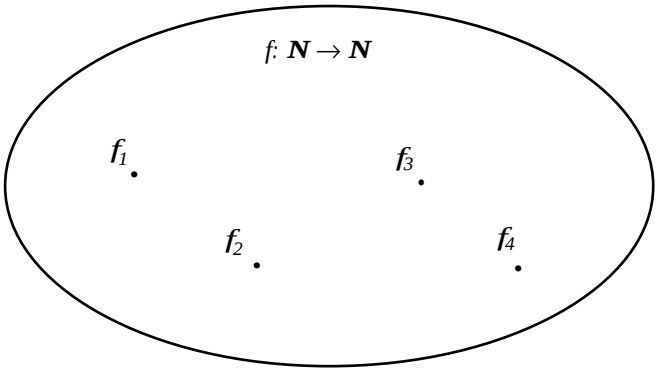

Figura 4.3 - Rappresentazione grafica di tutte le funzioni f: $\boldsymbol{N} \rightarrow \boldsymbol{N}$.

4.2 Funzioni calcolabili e non

A questo punto possiamo formulare in modo più preciso la domanda di inizio capitolo. Chiedersi "Ma con gli algoritmi si può fare tutto?" equivale a chiedersi "Data una qualsiasi funzione, esiste *sempre* un algoritmo che ne calcola i valori?". Chiamiamo *funzioni calcolabili* (o *computabili*) le funzioni con questa proprietà (esiste un algoritmo che ne calcola i valori). Ci stiamo quindi chiedendo: "Tutte le funzioni *f*: $\boldsymbol{N} \rightarrow \boldsymbol{N}$ sono calcolabili?". Bene la risposta è no: alcune funzioni sono non calcolabili. La situazione è rappresentata graficamente in figura 4.4.

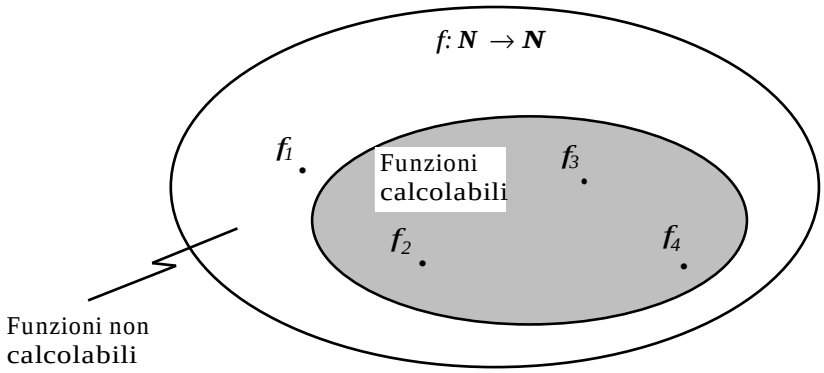

Figura 4.4 - Funzioni calcolabili e non.

Vediamo un esempio di funzione non calcolabile. Consideriamo il seguente problema (denominato *problema della terminazione*, o *halting problem*): "Trovare un algoritmo A' in grado di dirci se l'esecuzione di un qualsiasi programma A su dati D termina o continua all'infinito" (si veda la figura 4.5). Si rifletta su questo problema. Innanzi tutto, cosa significa che un algoritmo non termina? Ne abbiamo visto un esempio nell'esercizio 14 alla fine del capitolo 2: è sufficiente che alcune istruzioni vengano ripetute all'infinito.

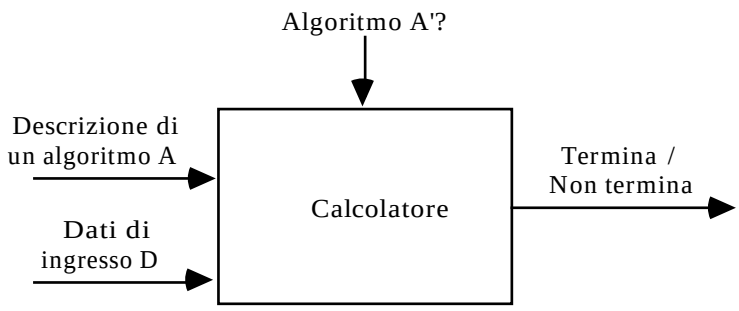

Figura 4.5 - Il problema della terminazione.

Proviamo a scrivere un tale algoritmo A'. Un primo tentativo potrebbe essere il seguente: far eseguire l'algoritmo A sui dati e attendere; se l'esecuzione termina allora la risposta è "sì", altrimenti la risposta è "no". Cosa c'è che non va? Beh, che la risposta "no" non può mai essere data (almeno non in tempo finito): se avviamo oggi l'esecuzione di A', quanti giorni aspettiamo prima di decidere che il programma non termina? E se il giorno successivo invece A' termina? Questa soluzione non funziona. E si può dimostrare formalmente (noi non lo facciamo) che un tale algoritmo A' non può esistere.

Ma se non tutte le funzioni sono calcolabili, allora esistono problemi non risolubili da un algoritmo? Ovviamente sì, il problema della terminazione ne è un esempio (si veda la figura 4.6).

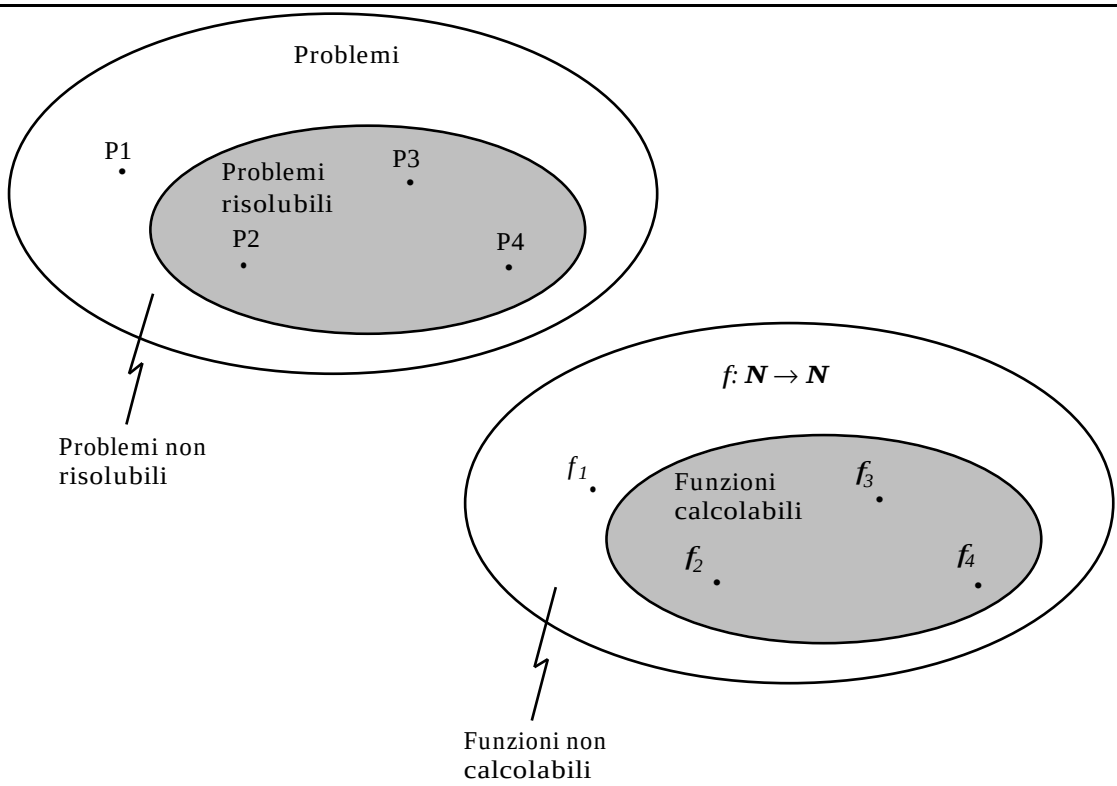

Figura 4.6 - Problemi risolubili e non.

A questo punto possiamo raffinare la figura 3.2, aggiungendo l'insieme delle funzioni (calcolabili) agli insiemi di problemi (risolubili), algoritmi e programmi. Il risultato è riportato in figura 4.7.

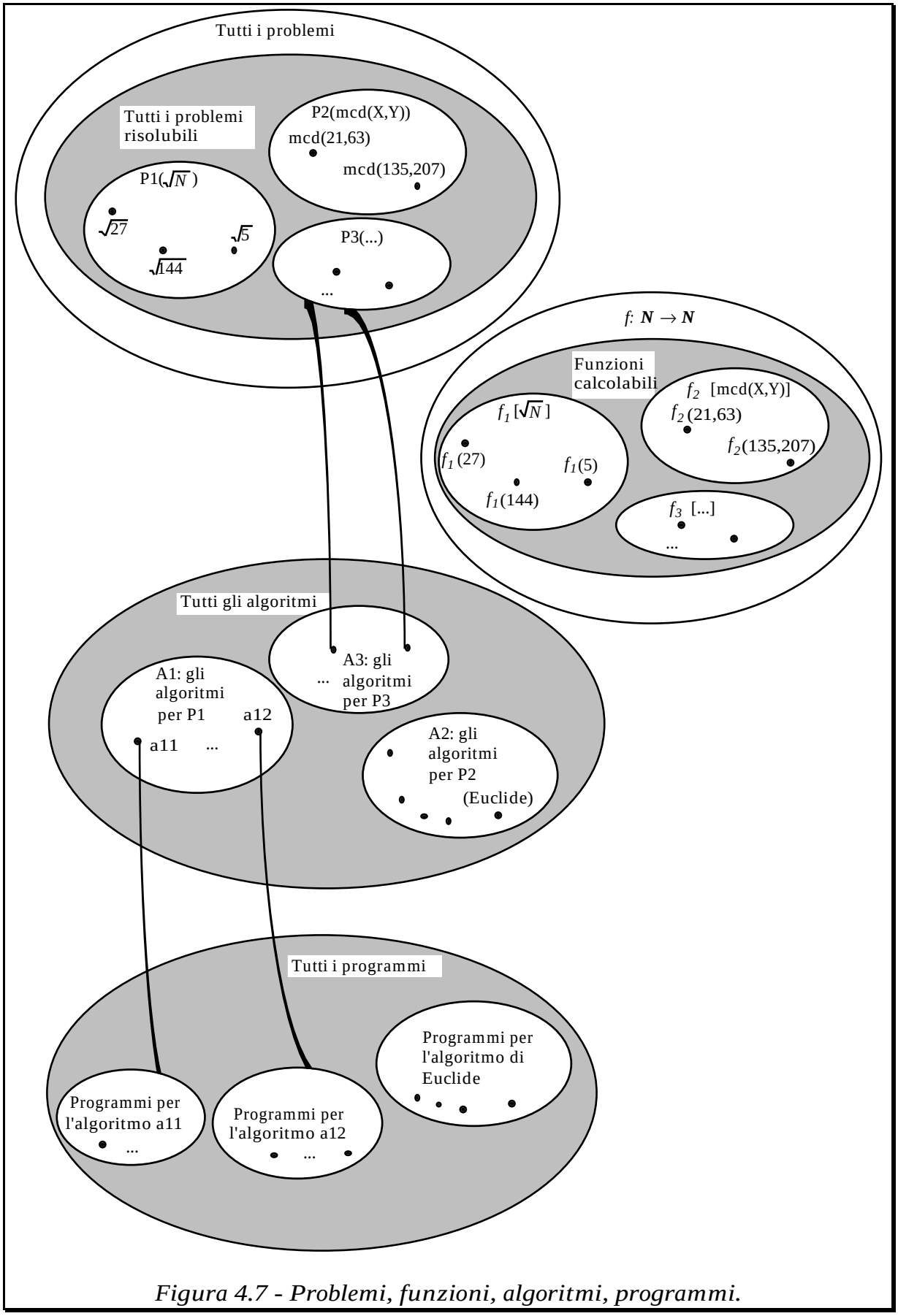

Figura 4.7 - Problemi, funzioni, algoritmi, programmi.

4.3 Tesi di Church-Turing

La seconda domanda posta all'inizio del capitolo era "Che cosa si può fare con gli algoritmi?". Ora possiamo riformularla in modo più preciso: "Quali sono le funzioni calcolabili?". La risposta viene dalla cosiddetta *Tesi di Church-Turing* (formulata negli anni trenta indipendentemente da due persone: Alan Turing e Alonzo Church): le funzioni calcolabili sono quelle *intuitivamente* calcolabili, ossia quelle che un essere umano è in grado di calcolare (si veda [Hof84, pagg. 606–625] per una discussione interessante, affascinante e anche divertente di questa tesi).

Questa risposta è un po' vaga, ma non si può fare molto di più: la nozione di funzione "intuitivamente" calcolabile è giocoforza vaga! Un modo più preciso per caratterizzare l'insieme delle funzioni calcolabili è quello di utilizzare la nozione di formalismo: noi possiamo dire che le funzioni calcolabili sono esattamente quelle che il nostro ufficio è in grado di calcolare. Ovviamente qualcuno potrebbe inventarsi un formalismo differente, e questo ci porta alla terza domanda di inizio capitolo.

4.4 Equivalenza dei formalismi

"Esistono algoritmi che sono rappresentabili in un formalismo e non in un altro?", o, in altri termini: "Ma non sarà che la nozione di calcolabilità dipende dal formalismo?", ovvero che l'insieme delle funzioni calcolabili varia a seconda del formalismo che utilizziamo? Noi abbiamo visto un solo formalismo (i diagrammi di flusso), e ne vedremo uno leggermente diverso nel capitolo 6 (i diagrammi di flusso strutturati), ma chi ci vieta di costruire un ufficio con un impiegato più potente, in grado di eseguire un insieme di istruzioni più articolato? Questa domanda non è banale come può sembrare a prima vista: se un formalismo è una restrizione di (o ne estende) un altro, non è detto che l'insieme di funzioni calcolate da quel formalismo sia più piccolo (grande). Si vedano gli esercizi 4.1–4.3.

Bene, anche qui non è possibile dare una risposta definitiva, ma c'è comunque un'argomentazione convincente. Nel passato (soprattutto intorno agli anni trenta) sono stati proposti parecchi formalismi, di natura piuttosto differente fra di loro: le macchine di Turing, le funzioni parziali ricorsive, i sistemi di riscrittura di Post, il lambda calcolo, le macchine a registri, e altri ancora. Ora, tutti questi formalismi sono *equivalenti*, nel senso che sono in grado di calcolare le stesse funzioni (quelle intuitivamente calcolabili, si riveda la tesi di Church). È quindi ragionevole aspettarsi che la nozione di calcolabilità sia catturata e che nuovi eventuali formalismi non "vadano più in là", ossia calcolino sempre lo stesso insieme di funzioni.

Il nostro ufficio è quindi in grado di calcolare tutte le funzioni calcolabili ed è un formalismo "completo", equivalente a quelli appena elencati.

4.5 Esercizi

Esecizio 4.1. Si modifichi l'ufficio aggiungendo l'operazione di elevamento a potenza fra le attività elementari che l'impiegato sa eseguire. L'insieme delle funzioni calcolate cambia? Perché?

Esecizio 4.2. Si modifichi l'ufficio eliminando le operazioni di moltiplicazione e di divisione. L'insieme delle funzioni calcolate cambia? Perché? (Suggerimento: si rifletta sull'algoritmo del paragrafo 2.3)

Esecizio 4.3. Si modifichi l'ufficio eliminando le operazioni di moltiplicazione e divisione e sostituendo l'operazione di somma con l'operazione di incremento ("+ 1") e l'operazione di sottrazione con l'operazione di decremento ("- 1"). L'insieme delle funzioni calcolate cambia? Perché?

Capitolo 5
La risoluzione dei problemi

5.1 La realizzazione di un programma

Il processo di risoluzione di un problema è rappresentato in figura 5.1, che estende la figura 3.3. Analizzando le esigenze dell'utente, il direttore le associa a un problema. A partire dal problema il direttore progetta l'algoritmo risolvente e lo rappresenta poi in termini di un linguaggio di programmazione comprensibile dal particolare calcolatore che ha a disposizione (l'ufficio). A questo punto il programma può essere eseguito sul calcolatore, utilizzando i dati di input forniti dall'utente, e i risultati dell'esecuzione vengono restituiti all'utente. La diversa forma dei blocchi in cui sono racchiusi le esigenze dell'utente, il problema, l'algoritmo e il programma evidenzia come il processo proceda da entità astratte e non ben definite verso oggetti più concreti e definiti in modo preciso.

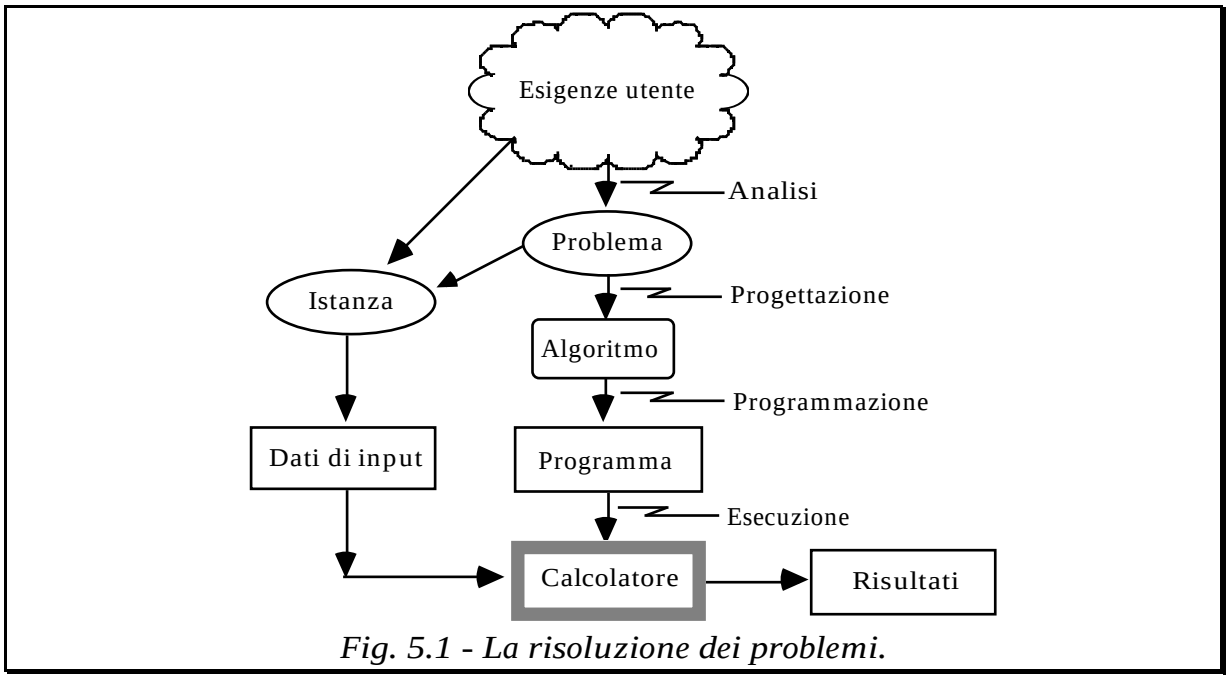

Fig. 5.1 - La risoluzione dei problemi.

Analizziamo brevemente le quattro fasi:

- *Analisi*. Il direttore (o *l'analista*, come viene detto in gergo informatico) deve analizzare le esigenze dell'utente, interagendo con lui e cercando di

fargli esplicitare i bisogni taciti. Di solito l'identificazione avviene analizzando le istanze del problema dell'utente, il quale è più affidabile se posto di fronte ad esempi concreti.

- *Progettazione*. Una volta individuato con la massima precisione possibile il problema dell'utente, il direttore deve progettare l'algoritmo risolvente. Si sente spesso dire che "la costruzione di un algoritmo è un'arte" e quindi è un'attività più simile a dipingere un quadro che a risolvere un'equazione o a svolgere un esercizio di analisi matematica. Per convincersene, si rifletta sul fatto che nel primo caso bisogna *costruire* una soluzione, mentre per risolvere un'equazione bisogna *trovare* una soluzione. Con l'aumentare dell'esperienza, il progettista si ritrova un bagaglio di conoscenze che può sfruttare (spesso un nuovo problema è simile a uno già risolto in precedenza).
- *Programmazione*. Una volta individuato l'algoritmo, non dovrebbe essere molto complicato tradurlo in un linguaggio di programmazione. Il problema principale è che spesso la descrizione dell'algoritmo prodotta nella fase di progettazione è ambigua, incompleta, talvolta contraddittoria o addirittura scorretta. E quindi il povero direttore (*programmatore*) si trova ad affrontare problemi non banali.
- *Esecuzione*. Finalmente il direttore (analista, progettista, programmatore) può rilassarsi e osservare il calcolatore al lavoro... ma ahimè spesso solo provando a eseguire il programma ottenuto su alcuni dati di prova (attività nota come *testing*, o collaudo) ci si rende conto che il programma non funziona. Bisogna quindi trovare le cause dell'errore (attività nota in gergo informatico come *debugging*) e (nei casi più sfortunati, ma non così rari) rifare tutto daccapo... Talvolta è il calcolatore a commettere errori, ma questa eventualità è molto rara.

Dovrebbe essere chiaro che non è l'impiegato a risolvere i problemi dell'utente. E anche l'ufficio, o calcolatore, gioca un ruolo marginale: il ruolo principale è quello del direttore, che deve svolgere un compito altamente creativo. Anche se ha un ruolo marginale, il calcolatore è comunque importante, siccome i calcolatori elettronici odierni sono estremamente:

- veloci: eseguono milioni di operazioni al secondo e il traguardo del *gigahertz* (un miliardo di operazioni al secondo) non è molto lontano;
- affidabili: talvolta anche i calcolatori commettono errori, durante l'esecuzione di un programma. Ma questo evento è estremamente raro, e tali errori sono facilmente identificabili;
- economici: un essere umano che deve svolgere gli stessi compiti di un calcolatore costa senz'altro di più, per non parlare di compiti pericolosi per gli umani. D'altra parte è innegabile che gli umani siano più flessibili dei calcolatori...

5.2 Gli errori

Il processo di costruzione di un programma, essendo un processo altamente creativo, è ricco di errori. Anche i programmatori più esperti difficilmene riescono a scrivere programmi completamente privi di errori. Ciò implica che il processo illustrato illustrato in figura 5.1 non è in realtà una semplice successione

di fasi in cascata. Capita spesso di ritornare indietro per rifare una fase precedente in cui era stato commesso un errore (o che non era stata terminata).

Gli errori possono manifestarsi in vari punti del processo illustrato in figura 5.1 e la loro gravità è tanto maggiore quanto più essi si verificano nelle fasi "alte" del processo. In tal caso è infatti possibile che l'errore non venga scoperto fino a quando si esegue una delle fasi "basse" e ciò può portare a dover rifare da capo buona parte del lavoro. È quindi importante procedere (almeno *il più possibile*: è pur sempre un'attività creativa) in modo *sistematico*, controllando i risultati intermedi. Ad esempio, una volta identificato il problema dell'utente (fase di analisi) è bene che il direttore interagisca con l'utente al fine di sincerarsi che tale identificazione sia corretta.

I prossimi paragrafi illustrano con maggiore dettaglio gli errori che si verificano nelle varie fasi della risoluzione dei problemi.

5.2.1 Analisi

Gli errori che si verificano durante la fase di analisi sono generalmente costituiti dall'identificazione inappropriata del problema, e quindi nella non corretta interpretazione delle esigenze dell'utente. Ciò è a sua volta dovuto a un'analisi incompleta, errata o superficiale delle esigenze.

Errori di questo tipo sono gravi poiché spesso si manifestano alla fine del lavoro o dopo un periodo d'uso durante il quale si esplicitano le carenze o le scorrettezze dell'analisi iniziale, e pertanto comportano il rifacimento di buona parte del lavoro, essendo necessario ripartire almeno parzialmente dalla fase di analisi.

5.2.2 Progettazione

L'errore tipico di questa fase consiste nella errata interpretazione dei risultati della fase di analisi, con la conseguente progettazione di un algoritmo che non rispetta le specifiche prodotte.

Spesso questo significa che alcuni sottoproblemi del problema principale non vengono compresi e quindi non vengono trattati dall'algoritmo. In questo modo, l'errore si manifesterà in esecuzione solo con determinate istanze del problema, rendendone difficile l'individuazione e l'eliminazione.

5.2.3 Programmazione

Durante la fase di programmazione sono possibili diversi tipi di errori, più o meno gravi. In particolare, l'errore più grave è quello determinato dall'implementazione errata dell'algoritmo progettato, che comporta la riscrittuta parziale o totale del programma.

Meno gravi sono gli errori determinati dall'uso scorretto del linguaggio di programmazione, sebbene anche essi siano di entità e portata diversificata; in particolare, ci sono errori (detti *statici*) che possono essere identificati da un esame statico del programma, e altri (detti *dinamici*) che invece si manifestano solo durante l'esecuzione. Alcuni degli errori statici sono usualmente individuati automaticamente dal programma di traduzione del linguaggio di programmazione (per esempio, gli errori sintattici nella scrittura del programma), altri invece necessitano dell'intervento del programmatore.

5.2.4 Esecuzione

Durante la fase di esecuzione gli errori possibili dipendono da malfunzionamenti del calcolatore, che opera diversamente da come ci si potrebbe aspettare; i malfunzionamenti possono essere determinati dall'hardware o dal software di base, e spesso sono facilmente identificabili. Questi errori sono comunque molto rari nei calcolatori moderni.

5.2.5 La gestione degli errori

Durante le fasi di analisi e progettazione, la correttezza dipende esclusivamente dal comportamento di analista e progettista, che possono seguire determinate metodologie al fine di ridurre la probabilità di errori.

Durante la fase di programmazione invece, come già accennato, buona parte degli errori dipendono dall'uso scorretto del linguaggio di programmazione, sebbene alcuni degli errori siano rintracciabili proprio grazie al linguaggio stesso.

Generalmente, il programmatore ha a disposizione almeno tre supporti all'attività di programmazione:

- gli strumenti di programmazione segnalano alcuni degli errori statici;
- l'esecuzione del programma durante la fase di test permette di evidenziare alcuni degli errori dinamici;
- il linguaggio di programmazione può avere caratteristiche formali tali da renderne più difficile un uso scorretto; i programmi di questo linguaggio, essendo chiari e leggibili, sono più facilmente verificabili staticamente.

Nel seguito, ci concentreremo sul terzo tipo di supporto, ovvero sulle caratteristiche che il linguaggio di programmazione deve avere per facilitare l'attività di programmazione.

5.3 I linguaggi di programmazione

Il linguaggio di programmazione è fondamentale nella costruzione di programmi, e non solo perché è il mezzo utilizzato per la realizzazione del programma stesso: le sue caratteristiche influenzano anche la fase di progettazione e in certi casi determinate scelte progettuali vengono compiute in vista delle peculiarità del linguaggio che si utilizzerà nella fase successiva. Infatti, sebbene tutti i linguaggi di programmazione permettano la soluzione degli stessi problemi, ognuno presenta punti di forza che lo rendono più adatto alla soluzione di certi problemi piuttosto che di altri.

Oltre a questo, il linguaggio di programmazione interviene anche nel "costringere" il programmatore a scrivere programmi chiari; a sua volta questo permette sia il suo controllo statico da parte del programmatore nel caso di errori, sia la verifica della sua corrispondenza all'algoritmo risolvente del problema iniziale (questo perché è necessario assicurare che un determinato programma sia effettivamente la soluzione di un problema specifico e non di altri). Ricaduta secondaria, ma non meno interessante, della leggibilità e chiarezza dei programmi è la loro riusabilità, che permette di riutilizzare programmi o loro parti per la soluzione di nuovi problemi, simili ai precedenti.

In generale dunque da un “buon” linguaggio di programmazione ci si attendono una serie di capacità, che rendano più efficaci le fasi di progettazione e programmazione:

- deve essere un aiuto concettuale in fase di progettazione;
- deve costituire una guida ai processi iniziali che permettono di passare dall’idea iniziale dell’algoritmo al programma vero e proprio;
- deve permettere la scrittura di programmi chiari, facilmente verificabili e facilmente modificabili.

Nel prossimo capitolo il linguaggio dei diagrammi di flusso verrà analizzato criticamente proprio da questo punto di vista, e verranno proposte delle soluzioni per i problemi incontrati.

Capitolo 6
I diagrammi di flusso strutturati

6.1 Troppa libertà

Il formalismo dei *diagrammi di flusso* (da qui in poi DF per brevità) utilizzato finora lascia la massima libertà al direttore: egli può combinare i blocchi di base con le modalità che preferisce, purché rispetti le regole del capitolo 2.

Questa libertà può sembrare positiva, ma non sempre lo è. Ad esempio, il direttore/programmatore può costruire DF molto contorti senza che ve ne sia un'effettiva necessità (magari, riflettendo un attimo, egli potrebbe trovare una soluzione più elegante e semplice per rappresentare lo stesso algoritmo).

Su DF particolarmente complessi, sarà quindi difficile:

- garantire la correttezza del programma (cioè garantire che risolva il problema per cui è stato realizzato);
- identificare gli errori;
- risalire dal programma al problema (utile se un altro programmatore lo osserva, o allo stesso programmatore dopo un po' di tempo...);
- modificare il programma se sopraggiungono nuove esigenze o per risolvere un problema simile.

Sarebbero desiderabili altri vincoli, che guidino il direttore nella costruzione di programmi più chiari. Introduciamo quindi questi vincoli; poi verificheremo se effettivamente ottengono l'effetto voluto e se per caso sono talmente restrittivi da impedire la rappresentazione di alcuni algoritmi.

L'introduzione di questi vincoli passa per tre punti fondamentali:

- individuazione di alcune configurazioni tipiche di blocchi che appaiono frequentemente nei diagrammi: sequenza, selezione condizionale e ciclo;
- costruzione delle regole che identificano tutti e soli i diagrammi che usano queste configurazioni;
- verifica che il nuovo linguaggio permette di risolvere gli stessi problemi.

6.2 La sintassi dei diagrammi di flusso strutturati

Un diagramma di flusso strutturato (DFS) è un DF composto con apposita regola e costruito come da figura 6.1, in cui il BSG (Blocco Strutturato Generico) può essere a sua volta una struttura di controllo o un blocco funzionale semplice, e ha un solo arco entrante.

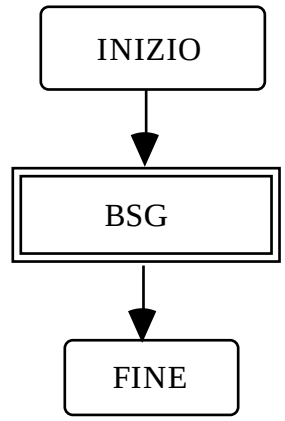

Figura 6.1 - Diagramma di flusso strutturato

Le strutture di controllo sono a loro volta costituite da tre possibili costrutti:

- *Sequenza*: identifica l'esecuzione sequenziale di due BSG, che a loro volta potrebbero essere sostituiti da altre sequenze (o altri BSG...) per ottenerne di più lunghe; rispetto alla sequenza DF che già conosciamo, non si prevede che altri archi puntino ai due BSG (figura 6.2).

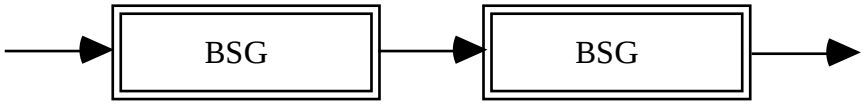

Figura 6.2 - Sequenza

- *Condizionale doppia*: definisce qualcosa di più preciso del solo blocco condizionale: scelta tra due azioni in base a una condizione logica (figura 6.3); se uno dei due BSG è vuoto, si ottengono i due casi particolari detti *condizionale semplice.*

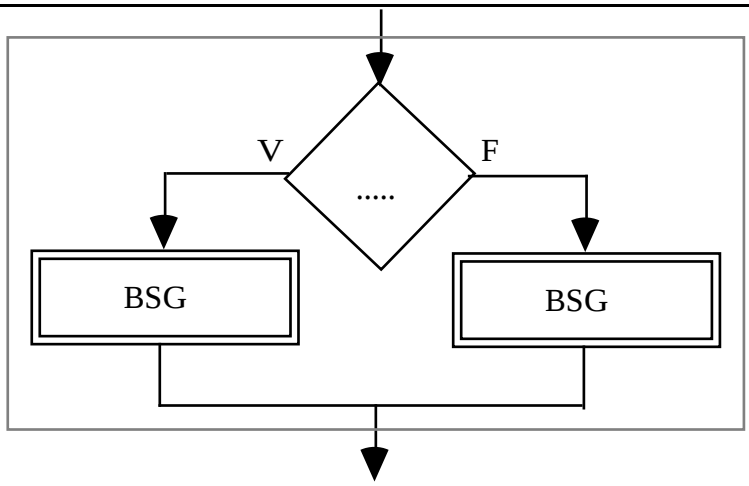

Figura 6.3 - Condizionale doppia

- *Ciclo a condizione iniziale*: il BSG viene ripetuto finché la condizione nel blocco decisionale è vera (figura 6.4); si noti che il BSG è eseguibile solo all'interno del ciclo (non ci si può arrivare tramite altri archi).

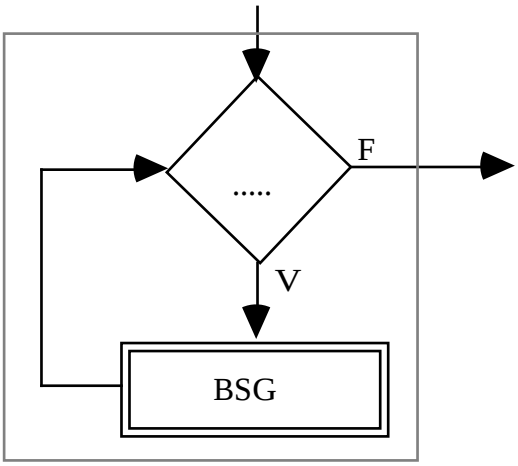

Figura 6.4 - Ciclo a condizione iniziale

Il BSG può anche essere un blocco funzionale semplice, contenente un'istruzione elementare, come nel linguaggio DF (figura 6.5).

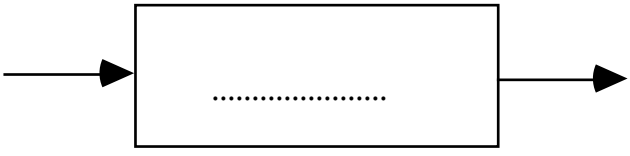

Figura 6.5 - Blocco funzionale semplice

Un diagramma di flusso strutturato si ottiene, a partire dallo schema iniziale, sostituendo per un numero arbitrario di volte un blocco strutturato generico con

- una struttura di controllo, oppure con

- un blocco funzionale semplice.

Si noti che sostituendo un BSG con una struttura di controllo si ottiene almeno un nuovo BSG da sostituire ulteriormente; mentre sostituendo un BSG con un blocco funzionale semplice, il processo si ferma.

Questo è anche il metodo che ispira la modalità di costruzione dei DFS, secondo il processo usualmente noto come *top-down*; sempre in base alla definizione, è possibile verificare se un diagramma sia strutturato o meno.

La figura 6.6 illustra un esempio di diagramma strutturato, ove le linee tratteggiate identificano i BSG. Le figure 6.7 e 6.8 mostrano invece dei controesempi, ossia dei DF che non sono DFS.

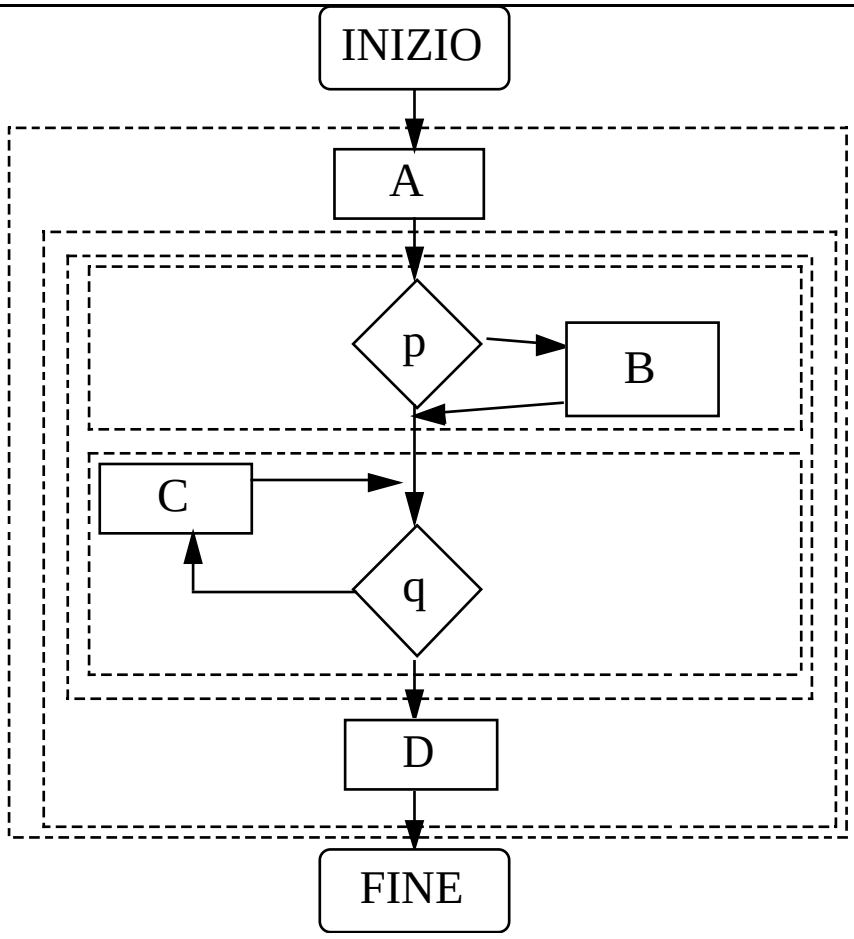

Figura 6.6

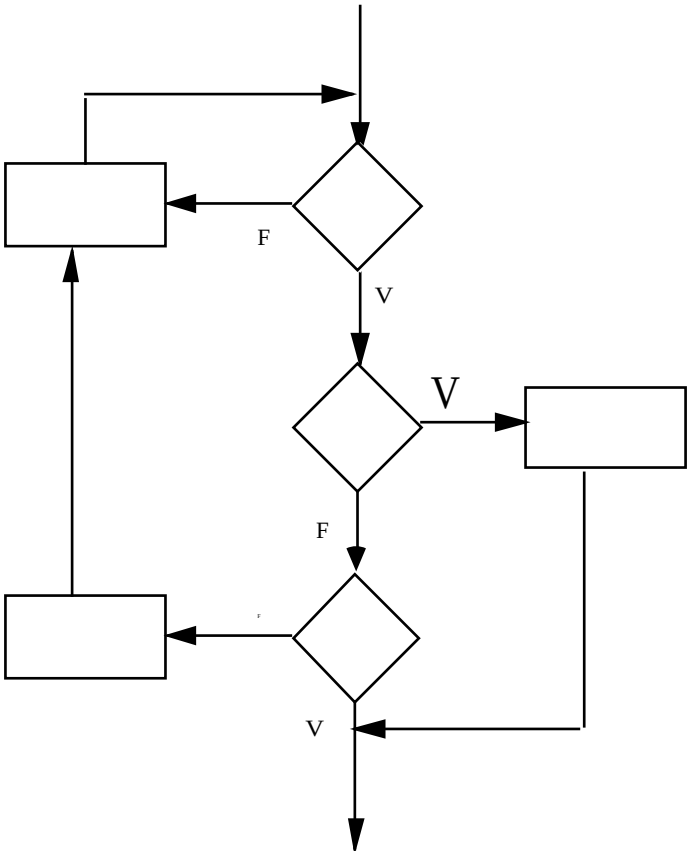

Figura 6.7

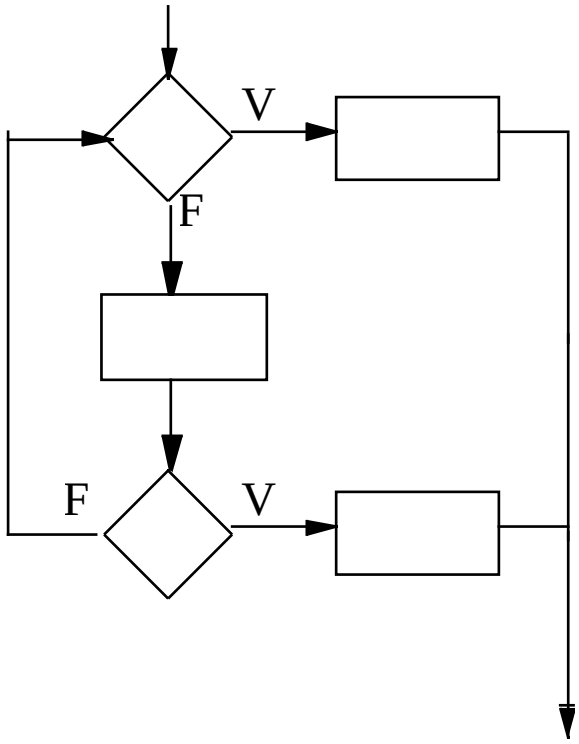

Figura 6.8

6.3 Confronto tra DF e DFS

Ora che abbiamo definito questo nuovo linguaggio DFS, si pone il problema di verificare se mantiene le caratteristiche desiderabili che già DF aveva, ovvero la possibilità di computare tutte le funzioni computabili, in tutti i modi possibili.

Questa verifica si compone quindi di due sottoproblemi:

- tutte le funzioni computabili con DF sono computabili anche con DFS? Cioè: tutti i problemi risolubili lo sono anche con DFS?
- tutti gli algoritmi esprimibili con DF sono esprimibili anche con DFS? Cioè: posso risolvere un problema risolubile con uno qualunque degli algoritmi risolventi anche esprimendolo in DFS?

Per effettuare queste verifiche, abbiamo bisogno di due concetti:

- *funzione computata* da un programma (con ovvio significato);
- *sequenza di computazione* generata da un diagramma di flusso: sequenza dei blocchi funzionali e decisionali generata dall'esecuzione del diagramma su dei dati iniziali. Fornisce il criterio di uguaglianza tra algoritmi.

In base a questi due concetti, possiamo definire degli strumenti per il confronto tra diagrammi generici:

- *Equivalenza debole*: due diagrammi di flusso X e Y si dicono *debolmente equivalenti* se e solo se, per ogni possibile dato d'ingresso, le funzioni da essi computate assumono lo stesso valore o sono indefinite.
- *Equivalenza forte*: due diagrammi di flusso X e Y si dicono *fortemente equivalenti* se e solo se, per ogni possibile dato d'ingresso, le sequenze di computazione generate sono uguali.

Ovviamente, l'equivalenza forte implica l'equivalenza debole.

A questo punto, i due problemi iniziali diventano rispettivamente:

- per ogni diagramma di flusso generico, esiste un diagramma di flusso strutturato debolmente equivalente?
- per ogni diagramma di flusso generico, esiste un diagramma di flusso strutturato fortemente equivalente?

La risposta alla prima domanda è data dal teorema di Böhm-Jacopini.

6.4 Il teorema di Böhm-Jacopini

Questo teorema, del 1966, fornisce sostanzialmente una risposta positiva alla prima delle due domande:

Per ogni diagramma di flusso esiste un diagramma di flusso strutturato debolmente equivalente.

Ciò significa che con i DFS possiamo computare tutte le funzioni computabili; la dimostrazione del teorema è di tipo costruttivo, e specifica il procedimento necessario per costruire un DFS debolmente equivalente a partire da un qualsiasi DF.

Per quel che riguarda il secondo quesito, la risposta è invece negativa:

non per tutti i DF esiste un DFS fortemente equivalente.

Ciò significa che posso computare tutte le funzioni, ma non per tutte posso scegliere l'algoritmo con la libertà che avevo con i DF.

La dimostrazione si ottiene mediante controesempio: non è possibile in alcun modo costruire un DFS fortemente equivalente al DF rappresentato in figura 6.9.

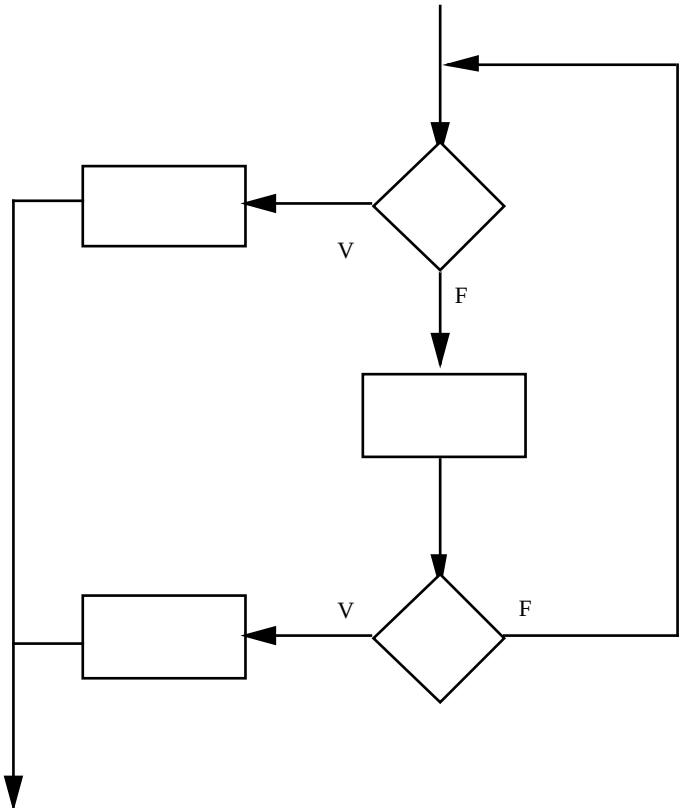

Figura 6.9

6.5 I limiti del linguaggio DFS

Anche se con DFS possiamo comunque computare tutte le funzioni computabili, abbiamo perso in potenza espressiva, non potendo più realizzare qualsiasi algoritmo. Come mai?

Possiamo dire che forse abbiamo introdotto delle regole troppo restrittive: i costrutti condizionale doppia e ciclo a condizione iniziale non catturano tutte le varianti rispettivamente di selezione e ripetizione, e infatti alcuni diagrammi presentano delle composizioni di questi tipo che non sono riscrivibili in modo strutturato.

La soluzione è introdurre delle regole leggermente più ampie che rendano più espressivo il linguaggio, in modo non solo di risolvere qualsiasi problema, ma di utilizzare anche uno qualsiasi degli algoritmi possibili.

6.6 Una soluzione: i Diagrammi di Flusso Ben Formati (DFBF)

Una soluzione è costituita da un linguaggio di diagrammi di flusso, chiamato DFBF (Diagrammi di Flusso Ben Formati) con delle regole analoghe ai DFS, ma un po' più generali. In particolare, in DFBF si hanno tre costrutti:

- *sequenza*, uguale a quella di DFS
- *ciclo generalizzato,* con un arco di ingresso e più archi di uscita governati da regole precise, che qui non riportiamo per semplicità.
- *blocco funzionale semplice*, come nei DFS.

Per diagrammi di flusso costruiti con queste regole, è stato dimostrato il teorema di Peterson-Kasami-Tokura (1973):

Per ogni diagramma di flusso esiste un diagramma di flusso ben formato fortemente equivalente

DFBF rispetta quindi i principi che avevano spinto verso DFS, ma senza limiti per gli algoritmi realizzabili.

Il ciclo generalizzato dei DFBF è comunque complicato da descrivere, e quindi i linguaggi di programmazione reali utilizzano delle vie di mezzo tra DF e DFBF, con costrutti aggiuntivi che rendono più semplice la scrittura dei programmi.

Capitolo 7
I calcolatori pensano?

[DA COMPLETARE...]

La domanda che dà il titolo a questo capitolo ha sollevato accesi dibattiti. Vediamo brevemente le argomentazioni più significative, con l'obiettivo di presentare brevemente alcuni spigolosi problemi fondazionali sollevati dall'informatica.

7.1 L'intelligenza artificiale

L'obiettivo dell'intelligenza artificiale è costruire programmi per calcolatori in grado di riprodurre le facoltà "intelligenti" dell'uomo. Siamo ben lontani da un risultato definitivo: abbiamo sì calcolatori che giocano a scacchi a livello dei grandi maestri, ma i calcolatori non sono in grado di riconoscere un viso o di giocare a tennis, e sono al livello di un mollusco nei rapporti con il mondo reale.

C'è chi sostiene che sia solo questione di tempo, dato che programmi "intelligenti" devono essere molto complessi, e quindi difficili da costruire. Altri sostengono che il problema principale è quello di dotare i calcolatori di un corpo, in modo da consentirgli di interagire con il mondo reale in maniera analoga a quello che facciamo noi umani. Altri discutono sulla correttezza di questa impostazione: la mente umana, essi dicono, non funziona "per algoritmi" e quindi è impossibile che un calcolatore riesca a comportarsi in modo intelligente.

Ma il primo problema da affrontare è ovviamente: come fare a dire se un calcolatore è intelligente?

7.2 Il test di Turing

Immaginate un esperimento di questo tipo: chiudiamo un calcolatore in una stanza e un uomo in un'altra. Entrambi possono comunicare con il mondo esterno unicamente tramite una tastiera e un monitor. Se non è possibile, ponendo domande tramite la tastiera e leggendo le risposte sul monitor, capire in quale stanza sta il calcolatore e in quale l'uomo, allora il calcolatore può essere definito *intelligente*. Questa è la proposta che Turing ha fatto nella prima metà del secolo.

In verità, Turing ha proposto questo *Test di Turing* in modo piuttosto scherzoso, ma è stato preso terribilmente sul serio da schiere di scienziati e filosofi che hanno adottato questo criterio per definire l'intelligenza. Da ciò sono

nate discussioni sulla simulazione e l'emulazione: un calcolatore emula l'intelligenza o la simula solamente? Una delle argomentazioni più note è quella illustrata nel prossimo paragrafo.

7.3 La stanza cinese di Searle

Lasciamo la parola al Filosofo John R. Searle [Sea90, pag. 16]:

> Supponiamo ora che io mi trovi in una stanza contenente scatole piene di ideogrammi cinesi e supponiamo che mi venga fornito un manuale di regole (scritto nella mia lingua) in base alle quali associare ideogrammi cinesi ad altri ideogrammi cinesi. Le regole specificano senza ambiguità gli ideogrammi in base alla loro forma e non richiedono che io li capisca. Le regole potrebbero essere di questo tipo: «Prendi uno scarabocchio dalla prima scatola e mettilo accanto alla seconda scatola».
>
> Supponiamo che fuori dalla stanza vi siano delle persone che capiscono il cinese e che introducano gruppetti di ideogrammi e che, in risposta, io manipoli questi ideogrammi secondo le regole del manuale e restituisca loro altri gruppetti di ideogrammi. Ora il manuale con le regole è il «programma di calcolatore», le persone che l'hanno scritto sono i «programmatori» e io sono il «calcolatore». Le scatole piene di ideogrammi sono le «basi di dati», i gruppetti di ideogrammi che mi vengono forniti sono le «domande» e quelli che io restituisco sono le «risposte».
>
> Supponiamo ora che le regole del manuale siano scritte in modo tale che le mie «risposte» alle «domande» non si possano distinguere da quelle di una persona di lingua madre cinese. [...] Io supero così il test di Turing per la comprensione del cinese, eppure ignoro completamente questa lingua. [...] Come un calcolatore, io manipolo simboli, ma non annetto a questi simboli alcun significato. [...] La sola manipolazione dei simboli non basta di per sé a garantire l'intelligenza [...]

(La "stanza cinese" di Searle ci ricorda qualcosa, vero?) Ma questa critica, a noi che abbiamo lavorato a lungo con il nostro ufficio, sembra debole: se non è l'impiegato che sa fare l'elevamento a potenza, ma tutto l'ufficio, allora non è l'"io" di Searle (il nostro impiegato) che deve capire il cinese, o essere intelligente, è tutta la stanza (ufficio), ossia l'insieme "io" + manuale + scatole di ideogrammi.

Critiche di questo tipo sono probabilmente deboli in partenza. Lasciamo la parola a Giuseppe O. Longo [Lon98, pag. 79]:

> Se il «pensiero» delle macchine avesse in tutto e per tutto gli stessi effetti che ha il nostro pensiero sul mondo, continuare a considerarlo un pensiero simulato sarebbe forse improprio, anche se qui si urta contro una resistenza psicologica [...], cioè di ordine diverso da quello logico. Alcuni considerano questa resistenza psicologica un pregiudizio e lo paragonano al pregiudizio che non vorrebbe concedere alle donne la facoltà di pensare: solo gli uomini pensano, le donne non pensano veramente, esse simulano il pensiero degli uomini, possiamo dire che

donna-pensano. Le lettrici allora si arrabbieranno? No, al massimo si donna-arrabbieranno e magari donna-getteranno via il libro.

7.4 Ma allora, pensano o no?

Insomma, non se ne esce. Proviamo con un ragionamento di tipo storico.

Nel medioevo la mente umana è stata paragonata ad altri artefatti, di gran lunga meno complessi dell'odierno calcolatore, ma fra i più complessi per quei tempi. Ad esempio, più o meno nello stesso periodo, la mente era paragonata in Germania all'orologio e in Gran Bretagna alla bilancia (e qui si potrebbe iniziare una lunga discussione sul carattere nazionale di quei due popoli...). Forse fra qualche secolo i nostri posteri si divertiranno alle nostre spalle così come noi ora ridiamo dei nostri antenati medievali. Ma forse no. Ovviamente, se si fa solo riferimento al passato, non ci si spiega il progresso: una critica analoga potrebbe essere stata fatta ai costruttori dei primi aerei... ma loro ce l'hanno fatta, a costruire gli aerei!

"Io di risposte non ne ho, io faccio solo rock and roll" diceva Edoardo Bennato. Forse questa è l'unica risposta certa (saggia?) che si può dare oggi.

Riferimenti bibliografici

[Bat84] G. Bateson. *Mente e natura*. Adelphi, Milano, 1984.

[Con98] L. Console e M. Ribaudo. *Introduzione all'informatica*. UTET, Torino, 1997.

[Dev91] K. Devlin. *Logic and Information*. Cambridge University Press, Cambridge, England, 1991.

[Göd31] K. Gödel. Uber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik, 38:173-198, 1931. Tradotto in *Il teorema di Gödel*, a cura di S. Shanker, Padova, Muzzio, 1991, pp. 23-62.

[Gui96] G. Guida. *Fondamenti di informatica - Algoritmi, programmi, sistemi di elaborazione*. Masson, Milano, 1996.

[Hof80] D. Hofstadter. *Gödel, Escher, Bach: un'Eterna Ghirlanda Brillante*. Adelphi, Milano, 1980.

[Lon98] G. O. Longo. *Il nuovo Golem. Come il computer cambia la nostra cultura*. Editori Laterza, Roma. 1998

[Neg95] N. Negroponte. *Essere digitali*. Sperling & Kupfer, Milano, 1995.

[Pen92] R. Penrose. *La mente nuova dell'imperatore*. Rizzoli, Milano, 1992.

[Pos93] N. Postman. *Technopoly*. Bollati Boringhieri, Torino, 1993.

[Pri93] I. Prigogine. *Le leggi del caos*. Fondazione Sigma-Tau, Lezioni italiane, Laterza, Roma-Bari (Italy), 1993.

[Sea90] J. R. Searle. La mente è un programma? In Le Scienze, n. 259, marzo 1990. pagg. 16-21.

[Sha48] C. E. Shannon. A mathematical theory of communication. Bell System Tech. Journal, (27):379-423, 623-656, 1948. http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

[SW49] C. E. Shannon e W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, 1949.

[Tur36] A. E. Turing. On computable numbers with an application to the entscheidungsproblem. In Proceedings of the London Mathematical Society, volume 42, pagg. 230-265, 1936.

[vB68] L. von Bertalanffy. *General System Theory*. Brazziler, New York, 1968.

[Wie48] N. Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*. Hermann, The Technology Press, Wiley, Paris, Cambridge, New York, 1948.