Il valore cognitivo dell'informatica: necessità di una impostazione sistematica

The cognitive value of computer science: the need for a methodical approach

Mario Fierli

Comitato per lo Sviluppo della Cultura Scientifica e Tecnologica del MIUR, mariofierli@virgilio.it

HOW TO CITE Fierli, M. (2017). Il valore cognitivo dell'informatica: necessità' di una impostazione sistematica. *Italian Journal of Educational Technology*, *25*(2), 66-69. doi: 10.17471/2499-4324/908

1. L'INFORMATICA NELLA SCUOLA: UN COMPITO DIFFICILE. SUPERARE GLI EPISODI E LE SOLUZIONI SEMPLICISTICHE

Dare all'informatica un posto adeguato nella formazione è difficile nella nostra scuola, come del resto nella maggior parte dei sistemi scolastici. Gli spazi specifici, quando ci sono, risultano sporadici, discontinui. Nei licei l'informatica compare come disciplina solo nell'indirizzo delle scienze applicate. Nell'istruzione tecnica, a parte l'indirizzo specialistico, le tecnologie dell'informazione e della comunicazione compaiono solo nei primi due anni. Aspetti dell'informatica sono collocati nelle più varie discipline-veicolo. La scelta più costante, a partire dal Piano Nazionale per l'informatica degli anni '80, è quella della Matematica. Nella scuola media questo compito è assegnato all'educazione tecnica, per altro in uno spazio orario molto angusto. Quando l'informatica è accolta all'interno delle diverse discipline, lo è in genere solo per il suo valore strumentale E nei suoi rari spazi specifici prevale l'obiettivo dell'alfabetizzazione digitale. Nella scuola primaria il problema non si pone in questi termini per il fatto che le discipline hanno confini meno netti.

Manca quindi storicamente una visione complessiva e verticale. Che non c'è neanche nei progetti sul Coding e nella legge 107 sulla "Buona Scuola" che affronta il problema in modo semplicistico. Il Piano Nazionale per la Scuola Digitale è molto più articolato, ma è ancora privo di un framework concettuale complessivo, la cui costruzione è prevista come uno dei risultati del progetto stesso, mentre, almeno in parte, dovrebbe esserne la premessa. Altri progetti, come il Problem Posing and Solving, propongono importanti obiettivi formativi, ma si tratta sempre di approcci parziali.

La costruzione di un framework generale che tenga conto di tutti gli aspetti cognitivi, contenutistici e metodologici è un compito a cui dedicare risorse. Va citato anzitutto il rapporto Developing Computational Thinking in Compulsory Education prodotto dal Joint Research Centre della Commissione Europea, che ha il compito di fornire supporto scientifico alle decisioni politiche (Bocconi, Chioccariello, Dettori, Ferrari, & Engelhardt, 2016). Negli Stati Uniti il K12 Computer Science Framework[1] che, come succede per tutti i più importanti progetti curricolari di quel paese, fornisce un impianto concettuale e operativo molto sistematico e autorevole, lasciando poi l'implementazione alle scuole e alle politiche degli stati, in mancanza di un curricolo nazionale.

Sarebbe anche utile che il MIUR adottasse la pratica, per tutti gli ambiti disciplinari e quindi anche per

[1] https://k12cs.org/

l'informatica, di fornire indicazioni di programmi di studio verticali, validi almeno per tutta la scuola dell'obbligo. E' quello che avviene nel Regno Unito con un programma di studio introdotto nel National Curriculum (Department for Education, 2013).

2. IL VALORE COGNITIVO DELL'INFORMATICA

È difficile far emergere il valore culturale e formativo dell'informatica, soprattutto a confronto con discipline (matematica, fisica, biologia ecc.) che hanno un grande e consolidato patrimonio epistemologico e pedagogico. Per questo è necessario lavorare per uno studio sistematico dei suoi aspetti cognitivi. Nel seguito di questo contributo si forniscono alcuni esempi ed elementi per la discussione.

2.1. *Informatica tra Problem Solving e pensiero computazionale*

Spesso si qualifica l'informatica come disciplina del Problem Solving. Occorre anzitutto distinguere fra l'attività di risoluzione di problemi in generale, che può assumere molte forme e molti aspetti cognitivi, con il Problem solving. Questo, secondo la psicologia cognitiva è la capacità di risolvere problemi non grazie a qualcosa che si è imparato, come i meccanismi psicologici istintivi o le regole e i metodi delle discipline, ma mediante procedimenti intuitivi-euristici. Il Problem Solving non è dunque un'attività teoretica, ma una manifestazione dell'intelligenza.

L'attività informatica è prevalentemente rivolta alla soluzione di problemi in generale, ma è sempre una forma di Problem Solving? Non lo si può affermare in modo incondizionato perché dipende dalle circostanze. Vale la pena di chiarire questo punto perché ha una rilevanza didattica. L'uso dell'informatica nel risolvere problemi introduce una sorta di sdoppiamento di livelli. Il primo livello è quello del problema originario, che non si vuole o non si può risolvere direttamente, e di cui si vuole affidare la soluzione a una macchina. Il problema si sposta a un secondo livello, il livello algoritmico. Quando i due livelli sono cognitivamente qualificabili come Problem Solving? Può succedere che il problema originario richieda una applicazione routinaria di regole, come semplificare un'espressione matematica, o addirittura una soluzione istintiva, come riconoscere il volto di una persona, mentre i corrispondenti problemi algoritmici sono invece difficili casi di Problem Solving o addirittura frontiere dell'intelligenza artificiale. Ma può succedere che il problema originario sia un caso di Problem Solving, come può capitare per la ricerca di un percorso in una rete. La soluzione del corrispondente problema algoritmico è un problem solving per chi non ha molte conoscenze su la teoria dei grafi e i relativi algoritmi ma non lo è per chi queste conoscenze le possiede.

Il concetto di *pensiero computazionale* è decisamente qualcosa di molto più vasto del semplice Problem Solving ed esce dalla sfera della psicologia cognitiva per entrare in quella della teoria della conoscenza. Jeanette M. Wing, nel suo oramai classico articolo (Wing, 2006), lo descrive come una vasta gamma di concetti e procedimenti che nascono dall'informatica (ad esempio la ricorsione, l'astrazione, la scomposizione, la rappresentazione), ma diventano una forma di pensiero applicabile tanto alla soluzione di problemi complessi quanto alla comprensione di molte realtà. Il pensiero computazionale dunque è esattamente l'opposto del pregiudizio, molto popolare, secondo il quale pensare in relazione alle macchine sia pensare come le macchine.

Il pensiero computazionale influisce in molti ambiti scientifici, come lo studio dei comportamenti umani, la biologia, l'economia, la fisica. Le diverse discipline possono incorporare concetti del pensiero computazionale, ma anche cambiare il loro modo di procedere. Il fisico Parisi, ad esempio, ha constatato che «*la vecchia dicotomia teorico-sperimentale è sostituita da una tripartizione: teoria pura (carta e matita), simulazione, esperimento*» (Parisi, 1991). Le discipline umanistiche e artistiche sono anch'esse coinvolte. Basta ricordare come Hofstatter, nel suo oramai classico libro (Hofstadter, 1984), fa muovere l'idea di ricorsione

fra la logica di Gödel, il disegno di Escher e la musica di Bach.
Il pensiero computazionale è la forma mentale indispensabile per *progettare e creare* sistemi informatici. Ma è anche lo strumento per *analizzare* quelli che già esistono e comprenderne il funzionamento. In che modo un telefono cellulare riesce a offrirci e correggerci le parole mentre scriviamo messaggi? Come funziona GOOGLE? Come si può descrivere il comportamento di un distributore automatico? Cioè è lo strumento per comprendere una gran parte del mondo fatto dall'uomo.

2.2. *Pensiero computazionale, programmazione e pensiero algoritmico*

Uno dei punti-chiave della Wing è che il pensiero computazionale non è riducibile alla programmazione e che l'appiattimento su questa ne fa perdere la complessità. In realtà la programmazione considerata nella sua interezza è pensiero algoritmico, non solo codifica, ed ha una sua ricchezza metodologica e cognitiva. A partire dagli anni '50, con lo sviluppo dei linguaggi di alto livello si è posta ad esempio la questione della struttura degli algoritmi e degli stili, o paradigmi, di programmazione. Alcuni linguaggi hanno collegato la pratica del programmare a principi teorici fondamentali come la calcolabilità e la ricorsività. Nella formazione, ci si è occupati molto anche del valore cognitivo dei diversi paradigmi di programmazione: quali sono più vicini al modo naturale di pensare gli algoritmi e/o favoriscono processi mentali più corretti? Il contributo più forte, sul versante educativo, è certo quello di Papert che ha proposto la programmazione in LOGO come strumento concreto al servizio di due idee fondamentali: quella dello studente-epistemologo e quella dell'apprendimento dei concetti secondo il costruttivismo piagettiano (Papert, 1980).
Una critica all'introduzione della programmazione nella scuola venne fin dagli inizi da Weizembaum che anticipò un argomento poi usato da molti: l'uso di modelli e la virtualizzazione inducono a una forma di ragione priva di complessità e lontana dalla realtà. E il semplicismo dei "piccoli" problemi-giocattolo spesso usati nella didattica non può dare una visione significativa del mondo reale (Weizembaum, 1976). Questa critica non è del tutto giusta perché non riconosce la funzione cognitiva in chiave costruttivista, anche di semplici programmi, nell'età evolutiva. Il Coding e il Making, che promuovono una cultura del fare e del risolvere problemi, per esempio, possono essere utili strumenti di sviluppo cognitivo, ma questo non avviene automaticamente: vanno inquadrati nella pedagogia del "bambino-epistemologo" illustrata da Papert. E comunque ai ragazzi più grandi, occorre proporre non problemi-giocattolo, ma situazioni progressivamente più complesse che permettano lo sviluppo di competenze significative e livelli più alti del pensiero computazionale.

2.3. *La soluzione di problemi informatici come costruzione di teorie*

Peter Naur, uno dei padri dell'informatica, membro, fra l'altro, del comitato per la creazione del linguaggio ALGOL, ha dato un grande contributo agli aspetti teorici e filosofici dell'informatica. In un saggio del 1985 Naur sostiene la tesi che la programmazione equivale alla costruzione di una teoria (Naur, 1985), partendo dall'affermazione che la cosa importante per un programmatore non è produrre codifiche o documenti, ma una teoria che spieghi come certi aspetti del mondo possono essere gestiti e sostenuti da un programma. Ferma rimanendo la funzione del pensiero computazionale nella soluzione di problemi, questa idea aggiunge ad esso un forte attributo culturale.
In effetti l'analisi dei sistemi che precede la programmazione e che usa modelli e mezzi di rappresentazione appare, indipendentemente dal suo valore pratico, un'attività teoretica. Ad esempio quando si usa un modello entità-relazioni per descrivere la struttura di un insieme di dati o una rete semantica, stiamo in qualche modo creando una teoria di quell'insieme di dati o di quel campo semantico.
Questo meccanismo cognitivo si può rivelare già a livello della creazione di algoritmi e della programmazione. Lo illustra in maniera efficace Papert quando racconta come uno studente affronta il problema di di-

segnare una circonferenza con il movimento della tartaruga del LOGO (Papert, 1980, p. 58). Lo studente si immedesima fisicamente nei movimenti della tartaruga, «*...per muoversi in cerchio si fa un piccolo passo avanti e ci si gira un poco e si continua così*» . Per arrivare al programma basta un passo:

PER CERCHIO
RIPETI
AVANTI 1
DESTRA 1

Questa è l'enunciazione di una legge matematica sotto forma di programma: una circonferenza è la curva ottenuta dal movimento di un punto che si muove a velocità costante deviandone in modo costante la direzione. Ed è, dice Papert, l'equivalente intuitivo di un'equazione differenziale.

3. BIBLIOGRAFIA

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing Computational Thinking in Compulsory Education - Implications for policy and practice*. EUR 28295 EN. doi:10.2791/792158

Department for Education. (2013, September 11). Statutory guidance National curriculum in England: computing programmes of study. Manchester, UK: Department for Education. Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Hofstadter, D. R. (1984) *Gödel, Escher, Bach: Un'eterna ghirlanda brillante*. Milano, IT: Adelphi.

Naur, P. (1985). Programming as theory building. *Microprocessing and microprogramming, 15*(5), 253-261. doi:10.1016/0165-6074(85)90032-8

Papert, S. (1980) *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books

Parisi, G. (1991) APE: un computer superveloce. In G. Cortini (Ed.) *Percorsi di fisica* (pp. 115-126). Firenze, IT: La Nuova Italia.

Weizembaum , J. (1976). *Computer power and human reason. From judgment to calculation*. New York, NY: W. H. Freeman & Co.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35. doi:10.1145/1118178.1118215