Consegne in itinere

Lorenzo Dentis, lorenzo.dentis@edu.unito.it

13 luglio 2023

Indice

1 Consegna 1 Domande parte introduttiva

1.1 Domande

Rispondere alle seguenti domande. Le risposte devono essere motivate citando le letture indicate su moodle. Dovete esprimere la vostra opinione rispetto alle posizioni espresse dagli articoli degli autori: siete d'accordo? se si/no perchè?

1. E' importante insegnare informatica come materia scolastica?
2. L'informatica è una scienza?
3. Qui sotto trovate una lista di criteri estratti dall'articolo The Science in Computer Science. Sono usati dall'autore per definire la credibilità di un settore come "scienza". Siete d'accordo con la scelta di questi criteri? Motivare le risposte:

   - Organized to understand, exploit, and cope with a pervasive phenomenon
   - Encompasses natural and artificial processes of the phenomenon.
   - Codified structured body of knowledge.
   - Commitment to experimental methods for discovery and validation.
   - Reproducibility of results.
   - Falsifiability of hypotheses and models.
   - Ability to make reliable predictions, some of which are surprising.

   L'informatica soddisfa qualcuno dei criteri sopra elencati? Se si quali? e perchè?

1.2 Risposte

1.2.1 È importante insegnare informatica come materia scolastica?

Secondo me insegnare l'informatica nelle scuole è importante, ma ancor più importante è insegnare le competenze digitali.
Ogni cittadino dovrebbe essere in grado di operare un computer o device digitale, perchè ci stiamo muovendo verso una società che per svoglere anche compiti necessare alla basilare sopravvivenza richiede di utilizzare dispositivi informatici. Saper utilizzare correttamente un computer e navigare su internet sta diventando una competenza basilare tanto quanto leggere e scrivere.

Parlando invece di *Informatica* invece credo che fornire delle basi sia importante.
Come l'autore di "Informatica e competenze digitali: cosa insegnare? "[5] , anche io credo che la necessità dell'insegnamento di questa disciplina derivi dal voler *"preparare i cittadini a comprendere appieno la società digitale"*. Trovo che l'Informatica come materia scolastica abbia molto in comune con la Storia, in quanto, anche nel caso in cui non rilevante nella crescita accademica di un individuo, è fondamentale per capire il mondo intorno a noi. Il manifesto per l'umanesimo digitale[3] parla molto di come le teconologie dovrebbero essere adattate a supporto della libertà e della democrazia, ma secondo me deve esserci anche una modifica del comportamento umano nei confronti della tecnologia. Non si può avere democrazia senza conoscenza. Anche presuppondendo di avere un prodotto perfetto e costruito con le migliori intenzioni, che rispetta la privacy dell'utente e garantisce imparzialità, tale strumento è inutile se il suo utilizzatore non si fida e non ci può essere fiducia se prima lo strumento non viene compreso. Un buon esempio può essere il voto elettronico. Se anche venisse realizzata una piattaforma online perfetta poi bisognerebbe convincere le persone ad usarla.Io stesso, come immagino chiunque altro, mi rifiuterei di votare se questa fosse una "black box" dal funzionamento sconosciuto.

Quindi concordo con l'affermazione *"L'educazione all'informatica e al suo impatto sociale devono iniziare il prima possibile"*, tratta dal manifesto sopra citato,perchè bisogna fornire ai futuri cittadini le competenze per utilizzare e sopratutto comprendere il funzionamento dei nuovi strumenti tecnologici che permeano tutti gli aspetti della nostra vita.

1.2.2 L'informatica è una scienza?

Si, come affermato durante le lezioni ciò dipende molto da cosa si considera scienza e cosa no. Io mi rifaccio alla definizione data da Galileo Galilei, cioè che una disciplina è scienza quando rispetta il metodo scientifico. Quindi una disciplina che segue le seguenti fasi: osservazione di un fenomeno, misura del

fenomeno, formulazione di una ipotesi, preparazione di un esperimento che provi l'ipotesi e che sia ripetibile. Nel caso dell'informatica ovviamente l'oggetto di studio è una informazione.
In particolare io "traccio la linea di demarcazione" tra scienza e non scienza sulla ripetibilità di un esperimento. Ad esempio non considero scienza le scienze sociali o l'economia, perchè gli esperimenti non possono essere effettuati "in ambiente sterile" ed i risultati ottenuti da differenti esperimenti non saranno mai identici, perchè è impossibile partire dalle stesse condizioni di partenza.
In informatica è invece partendo dalle stesse condizioni iniziali (che sono completamente controllabili) ed eseguendo lo stesso algoritmo si giunge sempre alle stesse conclusioni, l' informatica è deterministica.

1.2.3 Siete d'accordo con la scelta di questi criteri?

Organized to understand, exploit, and cope with a pervasive phenomenon.
Lo scopo della scienza è senza dubbio quello di dare una spiegazione ai fenomeni che avvengono intorno a noi.Gli altri 2 criteri "exploit and cope" trovo non appartengano tanto ad una disciplina scientifica quanto ad una disciplina *Tecnica*.Secondo me, basterebbe il primo criterio, senza includere gli altri due, d'altra parte questi sono conseguenza del primo.Una volta compreso un fenomeno è naturale che spontaneamente sorgano metodi per sfruttare questa conoscenza, altrimenti non ci si sarebbe interessati al fenomeno in primo luogo.

Encompasses natural and artificial processes of the phenomenon.
Anche riguardo a questo criterio non sono convinto, trovo sia irrilevante la natura del fenomeno.Non vedo perchè una disciplina che studi fenomeni artificiali non dovrebbe essere considerata scienza.

Codified structured body of knowledge.
Si, credo che ogni scienza per essere considerata tale abbia bisogno di basi, siano questi gli assiomi ed i teoremi della matematica o le varie teorie della biologia. Come poi è strutturato questo insieme di informazioni dipende da una scienza all'altra, ma senza una struttura diventa "una storia", letteratura piuttosto che scienza.

Commitment to experimental methods for discovery and validation.
Questo secondo me è il principale distinguo tra scienza e non scienza.Il metodo scientifico sviluppato da Galileo e la seguente elaborazione di Bacon sono a mio parere il punto di partenza per ogni scienza.Ad una ipotesi deve sempre seguire

una validazione logica e sperimentale, altrimenti non si può considerare provata una teoria.

Reproducibility of results.
Tale criterio segue dal criterio precedente, un esperimento per essere considerato rilevante deve essere ripetibile, come scritto nella domanda 1.2.2, non credo che le discipline che non soddisfano tale requisito siano definibili *scienza*

Falsifiability of hypotheses and models.
La scienza deve avere la possibilità di evolversi con il susseguirsi di nuove scoperte, non c'è posto nella scienza per concetti immutabili o "atti di fede".Anche le teorie più solide devono essere falsificabili.Come nel caso del famoso "paradosso dei corvi" [2], basta una controprova per rendere falsa una teoria.

Ability to make reliable predictions, some of which are surprising.
Mi interrogo sul perchè una scienza debba per forza ottenere risultati sorprendenti, dato che questo è un metro di valutazione soggettivo.Invece riguardo all'abilita di fare predizioni sono abbastanza convinto.Come accennato nel primo punto è una naturale conseguenza della comprensione di un fenomeno.

1.2.4 Quali criteri soddisfa l'informatica?

Organized to understand, exploit, and cope with a pervasive phenomenon.
Il fenomeno pervasivo che l'informatica studia è l'informazione e l'elaborazione della stessa.Non credo esista nulla di più pervasivo delle informazioni. Ogni fenomeno porta con se informazioni, dal comportamento umano alla biologia alla chimica.I metodi con cui noi umani abbiamo imparato a sfruttare la raccolta e l'elaborazione di informazioni sono abbastanza palesi, il primo che mi viene in mente è Internet.

Encompasses natural and artificial processes of the phenomenon.
Come dice il documento di Peter J. Denning[1] l'arogmentazione "l'informatica studia solo processi artificiali" cade quando si considera il fatto che l'informatica è la disciplina che studia le informazioni, non le tecnologie. L'articolo [1]invita anche a dibattere sul fatto che tutte le informazioni trattate dall'informatica derivano da fenomeni artificiali, anche le previsioni del tempo di un algoritmo metereologico non sono lo studio dell'evento metereologico in se, ma lo studio della simulazione dello stesso. Ciò non toglie che "le informazioni" non sono

per forza artificiali, anche una reazione chimica porta con se delle informazioni, quindi io credo che l'informatica soddisfi questo criterio, seppur non strettamente.

Codified structured body of knowledge.
La conoscenza dell'informatica è basata su principi teorici, modelli matematici e algoritmi che sono stati sviluppati e codificati nel corso degli anni. Basti pensare a tutti gli studi di informatica teorica quali algoritmi, complessità e calcolabilità, ma io includerei anche argomenti come i sistemi operativi, le reti, la sicurezza informatica, l'analisi dei dati, l'intelligenza artificiale e molti altri.

Commitment to experimental methods for discovery and validation.
Questo è un altro punto su cui si può dibattere, in quanto l'informatica di uso quotidiano non fa uso di esperimenti.Quando viene scritto un software difficilmente viene prima validato (nonostante ci siano strumenti per farlo).D'altra parte si potrebbe considerare l'uso da parte dell'utente come una sperimentazione, in quanto il software viene costantemente aggiornato, una specie di sperimentazione sul campo.
In ogni caso questa argomentazione decade quando si pensa a software sviluppato per scopo di ricerca.Mi viene in mente l'esempio del Q-learning, algoritmo di apprendimento automatico che si basa proprio sull'idea di effettuare molte azioni differenti e trovare quella che fornisce un risultato migliore in fase di addestramento, cioè esattamente sperimentare differenti strategie finchè nion si trova quella migliore.

Reproducibility of results.
Ho ampiamente discusso di questo punto nella domanda precendente 1.2.2.Partendo da un ambiente controllato, situazione molto semplice da produrre in informatica dato che siamo noi a generare gli input, gli algoritmi utilizzati produrranno output prevedibili.

Falsifiability of hypotheses and models.
Come ho accenanto in precedenza ci sono metodi formali per la validazione del software, che poi questi non vengano usati è un altro discorso.Il documento di Peter J.Denning fa lo stesso ragionamento (portando a supporto di questa tesi il fatto che circa il 50% dei modelli e delle ipotesi proposte non erano state rigorosamente testate).Direi che l'informatica soddisfa questo requisito se non

fosse per alcuni problemi indecidibili, primo tra tutti l'*halting problem* di Turing.In informatica (come in matematica) ci sono dei problemi indecidibili, di conseguenza non tutte le ipotesi ed i modelli sono verificabili falsi.Quindi a mio parere l'informatica non soddisfa questo criterio.

Ability to make reliable predictions, some of which are surprising.
Riguardo al fatto che l'informatica permetta di effettuare predizioni ho già speso molte parole.Sul fatto che siano sorprendeti mi sento di affermare che senza dubbio l'informatica permette di fare scoperte quantomeno non intuitive. I risultati delle scoperte informatiche hanno radicalmente cambiato il mondo forse più di quanto qualsiasi altra scienza abbia fatto.Un esempio di predizione sorprendente a mio parere è la teoria della complessità. Un problema semplice come le torri di Hanoi quando viene esteso sopra una certa soglia richiede un tempo computazionale sorprendente, cosa che non ci si aspetterebbe.

2 Consegna 2 Numeri Binari

2.1 Domande

Qual è il tema/concetto informatico oggetto dell'attività?

Quali sono gli obiettivi formativi?

Suddividete l'attività in fasi e per ogni fase individuate snodi e indicatori

Quali ingredienti delle varie teorie/metodologie viste nelle lezioni precedenti trovate in questa attività?

Riuscite a individuare nel testo dell'attività suggerimenti per il /la docente? secondo voi quali altre indicazioni devono essere integrate volendo rendere il documento una guida "completa" rispetto a snodi e indicatori?

2.2 Risposte

2.2.1 1 Qual è il tema/concetto informatico oggetto dell'attività?

Il tema è la codifica del dato in binario.Come rappresentare i numeri decimali in notazione binaria

2.2.2 2 Quali sono gli obiettivi formativi?

Io ho indivuato 4 obiettivi formativi

- O-P5-D-1. utilizzare combinazioni di simboli per rappresentare informazioni familiari complesse (es. colori secondari, frasi, ...);
  In questo caso i simboli sono prima le carte, poi i numeri binari.
- O-P5-N-3. comprendere come la riservatezza delle informazioni digitali può essere tutelata mediante codici "segreti";
  I numeri binari sono un esempio di codifica, per quanto basilare.Già questo può dare un idea di come "celare" dei dati, tale argomento viene approfondito soprattutto nella parte "extra", a pag 13.
- O-M-D-1. riconoscere se due rappresentazioni alternative semplici della stessa informazione sono intercambiabili per i propri scopi;
  Anche se è un obbiettivo da scuola secondaria credo sia presente. In questo caso però non viene evidenziato il parallelismo binario-decimale ne gli scopi
- O-M-D-2. effettuare operazioni semplici su simboli che rappresentano informazione strutturata (es. numeri binari, immagini "bitmap");
  Anche se è un obbiettivo da scuola secondaria questo obiettivo è decisamente presente. L'informazione è strutturata, per quanto la struttura sia decisamente semplice.

2.2.3 3 Suddividete l'attività in fasi e per ogni fase individuate snodi e indicatori

Fase 1, comprendere i Bit In questa fase vengono mostrate le carte agli alievi in modo da fargli comprendere le relazioni tra una carta (un bit) e l'altra.

- **snodi**
  - Comprendere il valore posizionale delle carte (ogni carta ha valore diverso in base alla posizione)
  - Comprendere la relazione tra una carta e la successiva (ogni carta ha valore doppio della carta alla sua destra)
- **indicatori**
  - Lo studente capisce che ogni carta ha il doppio dei punti della carta immediatamente alla destra
  - Lo studente è in grado di ipotizzare che valore avrebbe una nuova carta posta a sinistra dell'ultima.

Fase 2, comprendere il Byte In questa fase non consideriamo più la singola carta, bensì le 5 carte nel loro insieme (potremmo definirlo un byte).

- **snodi**

- Capire a che valore decimale corrisponde una precisa combinazione di carte.
- Essere in grado di passare dal valore numerico alla combinazione e viceversa.
- Comprendere la relazione tra numero di carte e possibili valori rappresentabili.
- Comprendere che ogni possibile numero (nel range consentito dalle carte) è rappresentabile in binario.

- **indicatori**
  - L'allievo è in grado di capire il valore rappresentato da una combinazione di carte.
  - Dato un valore l'allievo è in grado di girare le carte corrette.
  - L'allievo è in grado di dire qual'è il massimo valore rappresentabile con 5 carte e aggiungendo una carta come cambia.
  - L'allievo sa qual'è il minimo numero rappresentabile.
  - L'allievo è in grado di contare.

Fase 3, astrarre dalle carte In questa fase abbandoniamo le carte ed astraiamo ai numeri decimali.Questa fase comprende anche il foglio di lavoro a pag 13.

- **snodi**
  - Passare agevolmente da un numero decimale ad uno binario.
  - Comprendere che ci sono molti modi di rappresentare un numero binario, le carte o i disegni sono solo uno di questi. (in pratica comprendere che il binario, ed anche il decimale, sono solo un modo differente di rappresentare un numero)

- **indicatori**
  - L'alunno è capace di passare agevolmente dalla notazione decimale a quella binaria, senza l'ausilio delle carte.
  - Qualora gli sia presentato un numero binario in forma differente (i disegni nella parte extra) è in grado di rendersi conto che si tratta comunque di un numero.

2.2.4 4 Quali ingredienti delle varie teorie/metodologie viste nelle lezioni precedenti trovate in questa attività?

Costruttivismo, nella fattispecie l'apprendimento attivo, dato che ai bambini già dalla fase uno viene chiesto di interagire con gli strumenti per trovare una soluzione ad un problema. Citando Piaget nelle slides *i processi cognitivi tendono verso la viabilità, servono al soggetto per organizzare il mondo esperienziale e non per scoprire una realtà ontologica oggettiva.* In questa attività per poter rispondere alla domanda "che valore avrebbe una nuova carta posta a sinistra dell'ultima?"lo studente deve aver compreso che ogni carta ha il doppio del valore della carta precedente.
Oltretutto l'insegnante ha una influenza marginale sulla comprensione, il bambino impara autonomamente tramite una esperienza diretta con lo strumento (le carte), si verifica un processo di **scoperta attiva**.

Nell'attività proposta vi è anche un po' di Socio-costruttivismo, in quanto nelle prime 2 fasi gli studenti collaborano per giungere ad una risposta (ad esempio vengono scelti degli studenti per reggere le carte mentre altri cercano risposte alle domande) ed in generale è incentivato il **collaborative learning** ed il confronto.

Riscontro inoltre la presenza della strategia **Learning Cycle Instructional Models (5E)**, anche se mancano alcune fasi, come la valutazione.In ogni caso si rifà molto al processo descritto da Piaget ed analizzato nel documento *The Gears of My Childhood* [6] in cui le conoscenze vengono trasmesse incrementalmente, poggiandosi su basi pregresse.

2.2.5 5 che attività vengono suggerite al docente? come integrarle?

Le attività suggerite al docente sono:

1. Portare 5 studenti alla cattedra e formure ad ognuno una carta, poi cercare di far comprendere al resto della classe la regola che definisce il valore delle carte
2. Chiedere alla classe di fare ipotesi sul valore di carte successive
3. Scrivere diversi numeri in binario coprendo o scoprendo le carte
4. Chiedere agli studenti di fare lo stesso
5. Chiedere agli studenti il minimo numero rappresentabile e successivamente contare
6. Chiedere agli studenti di convertire da binario a decimale e viceversa senza l'ausilio delle carte

7. Passare ad altre rapresentazioni (spunte, picche, cerchi,etc..).

Io integrerei qualche attività/domanda per far comprendere meglio il fatto che il valore della carta dipende dalla sua posizione, cioè non posso mettermi a spostare le carte come mi pare.
Ad esempio al posto degli studenti (che si possono muovere) fisserei le carte su una bacheca e chiederei agli studenti di coprirle/scoprirle con un foglio.

L'altra cosa che farei è dedicare più tempo al conteggio, contando dal valore minimo (0) al valore massimo (32), facendoglio notare che non è possibile rappresentare un numero maggiore. Cioè renderei subito chiaro qual'è il range di valori rappresentabili e come incrementarlo (aggiungere una carta).

3 Consegna 3 Algoritmo

3.1 Domande

Tenendo conto delle discussioni svolte a lezione sulla definizione di algoritmo e delle sue proprietà scrivete una vostra definizione di algoritmo e delle sue proprietà motivando eventuali differenze rispetto alla formulazione vista a lezione. Nello svolgere questo compito tenete presente che la discussione sulle definizioni deve essere rivolta a classi delle scuole secondarie di secondo grado senza prerequisiti particolari.

3.2 Risposte

3.2.1 Definizione di algoritmo

Io ricordo che algoritmo mi fu definito come "una sequenza finita di istruzioni elementari per giungere da un problema ad una soluzione". Ad una ipotetica classe delle scuole secondarie di secondo grado riproporrei questa definizione, perchè riesce ad essere sufficientemente precisa ma è soprattutto molto succinta e facile da memorizzare e comprendere (tanto che ancora adesso la ricordo). Certo non è la definizione più esaustiva e formale, però nella sua semplicità mi piace molto.
Volendo andare nel dettaglio, e basandomi sulla discrussione in classe, definirei un algoritmo come *Una sequenza finita e ordinata di istruzioni elementari univoche che da un problema porta alla sua soluzione.*
Non è molto differente dalla definizione data in classe, secondo me è un po' migliore in quanto mette in evidenza il fatto che l'ordine di esecuzione delle istruzioni è molto importante, in secondo luogo credo che questa definizione sia più esplicita riguardo al fatto che l'algoritmo non è la soluzione, l'algoritmo associa un problema alla sua soluzione.

In questo sono della stessa idea del mio collega Cacioli (di cui ho sentito le argomentazioni nelle videoregistrazioni), secondo la mia interpretazione un algoritmo non deve essere corretto per essere considerato tale.Un algoritmo sbagliato è comunque un algoritmo, per due motivi.

1. Per quanto sia possibile verificare formalmente un algoritmo tale operazione non viene quasi mai svolta, quindi potrebbero esserci degli *"edge cases"* che vanno a vanificare la correttezza di un sacco di algoritmi.Probabilmente tali casi non ci sono, o sono rarissimi, ma senza una verifica formale non si può affermare che un algoritmo sia veramente corretto e basare la definizione di algoritmo su un assunto così forte mi sembra un po' troppo stringente. (Ricordo che il professor Roversi nel corso di Algoritmi e complessità in maniera provocatoria affermava: "Voi avete formalmente dimostrato che il quicksort fornisce una lista ordinata di elementi?No, vi basate sulla fiducia.Siete sicuri che qualcuno lo abbia mai dimostrato?")

2. Volendo essere molto precisi quando si sta parlando di **un** algoritmo ci si riferisce ad un generico algoritmo, non **all'** algoritmo specifico che risolve il problema specifico.Quindi l' algoritmo che non risolve il problema a mio parere è comunque un algoritmo.Non risolve quel problema, magari potrebbe addirittura risolverne un'altro (la professoressa stessa faceva l'esempio dell'algoritmo per la risoluzione di somme a due decimali, che però non funziona a n decimali).
   Quindi volendo includere la correttezza tra le proprietà secondo me bisognerebbe specificare che questa è la definizione di *un algoritmo risolutivo per il problema x.*
   Naturalmente mi rendo conto che questo secondo punto è veramente un esempio del "voler cercare il pelo nell'uovo" ciononostante credo sia una valida argomentaizione.

3.2.2 Proprietà di un algoritmo

- Finitezza: l'algoritmo deve terminare dopo un numero finito di passi

– io suddividerei in 2 parti la proprietà di **Precisione**.

  – Univocità delle istruzioni: Un algortimo deve essere composto da istruzioni non equivoche.

  – Semplicità delle istruzioni: Le istruzioni devono essere semplici ed eseguibili dall'interprete per cui viene scritto l'algoritmo.

- Dati in ingresso e in uscita: Un algortimo deve accettare zero o più input e deve fornire uno o più output che devono essere dipendenti dagli input immessi.
- Fattiblità: L'algoritmo deve essere implementabile, non ci possono essere operazioni astratte e non eseguibili.
- Efficienza: I passi dell'algoritmo devono tutti concorrere alla soluzione del problema, non ci devono essere passi che non avanzano la computazione. Nota, ciò è diverso da dire che un algoritmo per essere tale deve essere ottimizzato o efficiente, secondo me anche un algoritmo pessimo è un algortimo, basta che non faccia operazioni inutili.

Come discusso in precedenza io non inserirei la **correttezza** tra le proprietà di un algoritmo.

4 Consegna 4 Progettazione soluzione/algoritmo

4.1 Domande

- Leggete l'articolo Programming Patterns and Design Patterns in the Introductory Computer Science Course e scrivete un breve resoconto sull'approccio proposto:siete d'accordo con l'approccio proposto? lo adottereste? immaginate di proporlo in una classe in cui insegnate: quali sono le potenziali difficoltà?
- riportate il testo del problema analizzato per l'attività 2) Progettazione soluzione/algoritmo e la soluzione che avete progettato: suddivisione in sottoproblemi,pattern algoritmici, elementari e ruoli delle variabili.
- descrivete anche eventuali critiche (es. sulla formulazione del problema), problemi e considerazioni emerse durante l'attività

4.2 Risposte

4.2.1 Commento sull'articolo

In generale, concordo molto con il metodo presentato nell'articolo [7], anche perché è simile al modo in cui io stesso preferisco imparare nuovi argomenti. Nella sezione 4.2, subito dopo *Intent and Motivation*, viene presentata la sezione *Problem Examples*, in cui sono mostrati alcuni esempi di utilizzo del pattern. Trovo molto utile vedere prima un esempio di soluzione in atto quando si affronta un nuovo problema.

In futuro, gli studenti potranno far riferimento a questi pattern quando si troveranno di fronte a problemi simili, conoscendo già una possibile soluzione. Inoltre, questo metodo favorisce il riuso. Personalmente, durante il mio secondo anno di università, ho dovuto analizzare un file CSV molto complesso in Java e ho scritto del codice che ancora oggi utilizzo ogni volta che mi trovo ad avere in input un file con tale estensione.

Tutti i pattern presentati nell'articolo sono molto comuni, tanto che, prima di averli letti, li avevo già inseriti nella mia prima bozza dell'attività 2. In modo inconscio, ho scritto una domanda che richiede di risolvere due dei problemi presentati nell'articolo, volendo presentare un problema "semplice" e "comune".

In aula, una collega ha sostenuto che questo metodo potrebbe ostacolare l'apprendimento del *problem solving* e, in generale, concordo con questa affermazione. Cercare di categorizzare ogni problema e fornire la soluzione agli studenti rende la soluzione molto "meccanica" e non aiuta a sviluppare la capacità di "cercare" un modo di risolvere il problema.
Gli autori cercano di guidare gli studenti per evitare la situazione "I do not even know where to start", ma spesso capire da dove partire per risolvere un problema è una skill a sé, un'abilità che va sviluppata e che risulta utile nella vita quotidiana di uno sviluppatore.

Tuttavia, l'articolo non propone questo metodo come una panacea per insegnare a tutti gli sviluppatori, ma, citando ancora la sezione 4.2, *"The patterns are written to guide a novice with little or no programming experience"*. Quindi, si cerca di insegnare a programmare a persone con scarsa esperienza, che non hanno ancora le basi per iniziare a sviluppare la capacità di "cercare da soli una soluzione". In questo senso, credo che questo metodo sia eccellente.

4.2.2 Attività 2

Problema posto da Dentis `Lo Chef Tony vuole che la sua cucina sia sempre in ordine e soprattutto sapere quali ingredienti ha a disposizione. Quindi ha suddiviso gli ingredienti in 5 categorie:` *`pasta,sugo,carne,verdure e spezie`*`. Quando arriva un nuovo carico di materie prime (il numero di ingredienti in un carico può variare) Tony vuole che queste vengano separati automaticamente nelle 5 categorie e gli venga restituito in output quanti elementi di ogni categoria ci sono nel carico appena giunto.`

Il problema può essere complicato come nella seguente variante: `Il programma deve tenere conto che Tony non è l'unico Chef di Cat&Ring, nella dispensa`

potrebbero esserci ingredienti rimasti, quindi deve chiedere allo Chef la quantità rimasta e comportarsi di conseguenza.

Algorithm 4.1: La cucina di Chef Tony (pt.1)

Soluzione del problema posto da Dentis

```
1  input: int n, list carico
2  output: int pasta, int sugo, int carne, int verdure, int spezie
3  begin
4    i ← 0
5    pasta ← 0
6    sugo ← 0
7    carne ← 0
8    verdure ← 0
9    spezie ← 0
10   while i ≤ n
11      switch (categoria del prodotto i-esimo presente nel carico)
12        case pasta:
13            pasta ← pasta + 1
14        case sugo:
15            sugo ← sugo + 1
16        case carne:
17            carne ← carne + 1
18        case verdure:
19            verdure ← verdure + 1
20        case spezie:
21            spezie ← spezie + 1
22      i ← i + 1
23   end
24   return pasta, sugo, carne, verdure, spezie
25 end
```

Pattern elementari

- Alternative Actions: riga 11 (esistono x casi diversi, con x azioni diverse da intraprendere)
- Process-All-Items: riga 4 (in quanto ogni elemento della lista viene selezionato, processato e, poi, ci si prepara per il successivo oggetto)

Pattern Algoritmici

- Traversal Pattern: in particolare, si tratta dell'utilizzo di un *simple linear traversal pattern* (in quanto stiamo scorrendo lungo una lista di elementi)
- Cumulative Result Patterns: soprattutto vista la definizione del goal data nell'articolo *Programming Patterns and Design Patterns in the Introductory Computer Science Course*, che rappresenta in pieno l'approccio utilizzato durante la stesura dell'algoritmo: *the goal is to traverse some collection of data, collecting partial information into some accumulator entity and presenting the composite result at the end*
- epetition Pattern: in particolare una *conditioned repetition*, in quanto è presente un ciclo while all'interno dell'algoritmo

Ruoli delle variabili

- n: ha il ruolo di ***valore fissato***; rappresenta, infatti, la dimensione della lista carico che non viene a modificarsi durante lo svolgimento dell'algoritmo (volendo potrebbe essere sostituito da *carico.length* che esprime la lunghezza della lista carico)
- i: ha il ruolo di ***contatore***; rappresenta, infatti, lo scorrimento della lista *carico* mentre si modificano le variabili riferite ai tipi di alimenti presenti
- *pasta, sugo, carne, spezie, verdure*: hanno il ruolo di ***accumulatori***; rappresentano, infatti, l'accumulo di tutti i valori trovati fino a un momento i-esimo di una certa categoria di elementi

Problema posto da D'Angelo `Tra meno di 30 giorni, si svolgerà a Liverpool la sessantasettesima edizione dell'Eurovision Song Contest! Vista una serie di problemi avvenuti nelle votazioni dello scorso anno Martin Österdahl, produttore esecutivo della manifestazione, ha deciso di tenere sempre una copia della classifica in un pc separato e inattaccabile da hackers.`
`Il tuo compito è, dato il singolo paese, vedere i punteggi che quella nazione assegna ad ogni altro partecipante in gara e restituire la classifica completa.`
`I punteggi che possono essere assegnati sono:`

- 0 (default)
- 1 ¡= punteggio ¡= 8
- 10
- 12

Algorithm 4.2: Calcolo punteggi

Soluzione del problema posto da D'Angelo

```
1   input: lista_voti (che paese ha votato ogni paese).
2   output: classifica
3   begin:
4   num_paesi ← lunghezza(lista_voti)
5   classifica ← vettore[num_paesi] //vettore con tutti i paesi ed i voti ricevuti
6   foreach voto in lista_voti
7       foreach paese in classifica
8           if voto uguale paese
9               classifica[paese] ← 12 + classifica[paese]
10              break
11          end
12      end
13  end
14  return classifica
```

Pattern elementari

- Guarded-Action: riga 8
- Process-All-item: riga 6, scorriamo tutta la lista dataci in input
- process-Items-Unitl-Done: riga 10. Mentre si sta scorrendo il vettore della classifica se si trova il paese a cui bisogna assegnare il voto si può interrompere la ricerca.

Pattern Algoritmici

- 3.7 Traversal Pattern. In particolare *simple linear traversal* dato che stiamo scorrendo due vettori

- 3.5 Repetition Patterns. Dato che abbiamo due loop annidiati.
- 3.8 Cumulative Result Patterns. usiamo una composizione di 3 pattern, due repetition pattern (i loop annidiati) ed un accumulatore (il vettore classifica che tiene conto dei punteggi). citando il documento [7] *he goal is to traverse some collection of data, collecting partial information into some accumulator entity and presenting the composite result at the end.*

Ruoli delle variabili

- Valore fissato: la variabile *num_paese* serve solo in fase di inizializzazione per fissare la dimensione della classifica. In realtà se ne sarebbe anche potuto fare a meno.
- Contatore o Indice: la variabile *paese* svolge anche il ruolo di indice del vettore $classifica$
- Accumulatore: la variabile $classifica$ è un vettore di accumulatori.

4.2.3 Considerazioni emerse

La versione originale del problema prevedeva una struttura dati differente: una collection di oggetti (paesi) contenenti una lista di voti. Ogni paese quindi avrebbe avuto una sua lista di punteggi assegnati ad ogni altro paese.

Il problema è che la struttura dati necessaria a rappresentare tale collezione è un po' troppo complicata per il target, dato che necessiterebbe la nozione di *object* oppure l'uso di una matrice bidimensionale.

5 Consegna 5 Misconception

5.1 Domande

Partendo dall'appendice A della tesi Visual Program Simulation in Introductory Programming Education.

1. Provare a classificare le misconceptions in base al tipo di difficoltà (sintattica, concettuale, strategica), scrivetene alemno 5 per categoria nella consegna.
2. Data la vostra esperienza (di studenti delle superiori, di studenti universitari, alcuni di voi come tutor, alcuni di voi come persone che fanno ripetizioni), provate ad elencare "misconception" nella programmazione in cui vi siete imbattuti, personalmente o in altri.
3. Come le avete risolte?

5.2 Risposte

5.2.1 1 Classificare misconseptions

Misconception **sintattiche**:

1. **[10]**Variables always receive a particular default value upon creation.
2. **[38]**A return values does not need to be stored (even if one needs it later)
3. **[80]**An object is a subset of a class. / A class is a collection of objects.
4. **[125]**Cannot have methods with the same name in different classes
5. **[160]**Confusing textual representations with each other, e.g., the string "456" with the number.

Misconception **concettuali**:

1. **[8]**Magical parallelism: several lines of a (simple non- concurrent) program can be simultanenously active or known.
2. **[16]**Assignment moves a value from a variable to another.
3. **[47]**Subprograms can (routinely) use the variables of calling subprograms.
4. **[67]**Assigning to an object causes it to become equal to the assigned object.
5. **[124]**Objects 'know' which methods are operating on them (rather than method calls 'knowing' which object they operate on)

Misconception **strategiche**:

1. **[4]**The system does not allow unreasonable operations.
2. **[17]**The natural-language semantics of variable names affects which value gets assigned to which variabl
3. **[33]**loops terminate as soon as condition changes to false.
4. **[54]**Null model of recursion: recursion is impossible.
5. **[158]**Confusion between data in memory and data on screen.

5.2.2 2/3 Misconseption incontrate e come sono state risolte

11 Primitive assignment works in opposite direction. Probabilmente questa misconception deriva dalla matematica o dal metodo di scrittura "da sinistra a destra", però ricordo che quando ho iniziato a programmare trovavo più naturale che il valore venisse preso da sinistra e portato a destra dell'uguale. Questa misconception è stata di facile risoluzione in quanto una volta identificato cosa stavo sbagliando è stato solo una questione di ricordarmi "l'assegnamento va verso sinistra" o alla peggio lanciare il programma, realizzare il mio errore e girare l'assegnamento.

Una misconseption simile alla *22 Unassigned variables of primitive type (in Java) have no memory allocated.* ma diametralmente opposta mi è capitato di vederla nel corso di SO. Il collega sosteneva che si poteva allocare dinamicamente memoria come in java, senza bisogno di chiamate alla `malloc`.
Ad esempio tentava di fare la seguente operazione:

```
float read_and_process(int n)
{
    float vals[n];
    ....
}
```

Il motivo era che al posto di compilare da riga di comando con i parametri specificati dal professore (usando C90) il collega usava un IDE che di default aveva impostato C99. Per risolvere la misconception è bastato che il professore gli mostrasse il changelog di C99 in cui venivano introdotti i variable-length array e lo obbligasse a compilare da riga di comando.

Ma la misconseption più comune che mi è capitato di incontrare è stata la *1 The computer knows the intention of the program or of a piece of code, and acts accordingly.*Soprattutto nelle scuole superiori succedeva spessissimo che venissero giustificati errori nel codice con la frase "si ma io ovviamente intendevo questo" e la risposta del mio professore era sempre "il computer è una macchina stupida, si limita a fare ciò che gli dite di fare". Trovo che questa misconception sia la più pericolosa ed anche la più difficile da "risolvere" in quanto spesso durante la programmazione bisogna fare degli assunti sul funzionamento di un programma, una libreria o un sistema ipotizzando che il sistema si comporti in un certo modo piuttosto che un altro. Lei professoressa citava ad esempio il "perdere il filo" dello sviluppo durante il passaggio da una fase all'altra dello sviluppo software in SAS, questa cosa capitava costantemente. Ricordo che nel mio gruppo ricontrollavamo più e più volte di aver considerato tutto e comunque ci si rendeva conto di aver perso pezzi tra una fase e l'altra e si doveva tornare

alla fase precendente per aggiustare. Credo che questa misconception si risolva in due soli modi: tramite l'esperienza con l'uso di un sistema specifico/la ripetizione di una procedura (nel caso dello sviluppo software) e tramite la lettura della documentazione dove c'è scritto "il sistema fa questo e non fa quest'altro", operazione che richiede sempre un certo dispendio di tempo.

6 Consegna 6 la natura dei programmi

6.1 Domande

Partendo dal materiale relativo al seminario tenuto dalla Dott.ssa Violetta Lonati Di cosa parliamo quando parliamo di programmi scrivete una breve riflessione su questi aspetti:

- Nei corsi di studio che avete affrontato sono emerse tutte le sfaccettature del concetto di programma presentate nel seminario?
- Guardando le indicazioni nazionali del laboratorio Informatica e Scuola del cini e quelle degli istituti superiori vi sembra che includano tutte le 6 facce dei programmi?
- Dovendo progettare un intero corso di scienze informatiche in una scuola superiore in che ordine presentereste le 6 sfaccettature?

6.2 Risposte

6.2.1 1 Esperienze personali

Ho frequentato il liceo scientifico delle scienze applicate e ho avuto la fortuna di avere un buon insegnante di informatica. Abbiamo coperto quasi tutti i sei campi, ma ci siamo concentrati principalmente sui programmi come oggetti nozionali, approfondendo molto il linguaggio C ad esempio, e sui programmi come oggetti astratti, con molta enfasi sulla differenza tra algoritmo e codice e sul problem solving piuttosto che sulla scrittura di codice. È stato dato molto spazio anche al tema dei programmi come entità eseguibili, poiché abbiamo fatto molta programmazione.

Contrariamente a quanto si potrebbe pensare, non abbiamo visto molto il programma come strumento, ricordo pochi riferimenti alle competenze digitali o all'informatica in ambito interdisciplinare. Tuttavia, abbiamo utilizzato strumenti a supporto dello sviluppo di algoritmi o della scrittura di codice, come Scratch, Algobuild e vari IDE.
Non è stata quasi affrontata la visione del programma come opera dell'uomo, ad eccezione di una lezione in cui sono stati affrontati aspetti del ruolo dell'informatica nella storia, parlando di Alan Turing e della macchina Colossus. Non

è stato affrontato il tema del programma come oggetto fisico, nonostante una breve introduzione all'hardware, all'architettura e alle reti, ma mai in relazione ai programmi. Il professore ha però proposto una attività facoltativa durante una autogestione riguardo alla programmazione di microcontrollori (Arduino)

In ambito universitario sono state ampiamente approfondite tutte e sei le sfaccettature.

6.2.2 2 Indicazioni delle scuole

Gli obbiettivi ed i traguardi nel documento del CINI mi sembra affrontino tutte e sei le sfaccettature. Nella scuola primaria si nota una prevalenza di programma visto come **entità astratte** e come **strumento** ad esempio T-P-1, T-P-2 e T-P-11. Tutti gli obbiettivi raccolti sotto l' *Ambito creatività digitale* sono una visione di **programma come strumento**.
Si può dire che nella scuola primaria manchi la visione di programmi come **opere dell'uomo**, **oggetti fisici** ed **entità eseguibili** ma il motivo di tale carenza è probabilmente dovuto al fatto che questi concetti sono più astratti e difficili da afferarre per uno studente senza altre basi. Tale discorso non regge però per la sfaccettatura "opere dell'uomo", si potrebbe tranquillamente mostrare anche nella scuola primaria tutto il discorso di programma come derivato di un'analisi dei requisiti o delle scelte di chi il programma lo "produce".

Invece vedo che sia nelle scuole secondarie che negli istituti superiori vengono incluse tutte le 6 facce dei programmi, anche il programma visto come opera dell'uomo.

6.2.3 3 Ordine di presentazione

Dovendo scegliere proporrei questo ordine:

1. Entità astratte
2. Artefatti linguistico-notazionali
3. Entità eseguibili
4. Opere dell'uomo
5. Strumenti
6. Oggetti fisici

Il primo passo è capire cosa sia un algoritmo e come possa essere usato per risolvere problemi. È possibile utilizzare esempi di problemi, non necessariamente problemi reali, per illustrare il concetto.**Programma come entità astratta**. Successivamente, è possibile passare dal modello "pseudocodice" alla codifica,

utilizzando anche un linguaggio semplice come Scratch. Questo serve sia per introdurre un po' di azione pratica, che secondo il costruttivismo aiuta a ricordare meglio le nozioni, sia per mantenere vivo l'interesse degli studenti, fattore da non sottovalutare. **programma come artefatto linguistico-notazionale**.

Una volta introdotto l'algoritmo e un linguaggio di programmazione, si può pensare di eseguire il programma, anche presentando un debugger o una metodologia di logging, come delle print in mezzo al codice per comprendere lo stato delle variabili durante l'esecuzione. **programma come entità eseguibile**

Dopo aver compreso cosa sia un algoritmo e aver scritto programmi semplici, è importante presentarli come **opere dell'uomo** e spiegare che hanno senso solo nel contesto in cui risolvono un problema specifico e che soprattutto sono limitati secondo i requisiti. Si possono presentare problemi reali e strutturati e proporre programmi per la loro risoluzione, magari anche scritti dagli studenti, che possano essere utilizzati come strumenti. Un esempio classico è la calcolatrice che permette di risolvere equazioni di secondo grado con la formula del delta. Oppure si possono utilizzare programmi in ambito interdisciplinare, mostrando il programma come **strumento**. Infine, si può presentare la visione del programma come **oggetto fisico**, spiegando l'architettura hardware e alcuni concetti di come funzionano CPU e memoria, sempre con l'ausilio di esercizi. Ad esempio si può usare la manipolazione di stringhe in C per far vedere che queste non sono altro che sequenze di interi ad 8 bit posizionati in settori di memoria contigui delimitati da un carattere "\0".

Sviluppando questa risposta, mi sono reso conto di quanto sia utile in realtà portare avanti tutti gli aspetti in parallelo, almeno i primi 3-4. Mentre si presenta la struttura di un algoritmo, si può iniziare a mostrare i primi costrutti in codice, anche con esercizi di lettura del codice. Una volta acquisite le basi, si può già pensare di eseguire programmi, senza necessariamente completare i primi punti. Allo stesso modo, si possono fare accenni alla memoria, alla complessità o allo scopo di un programma, anche se gli studenti non sono ancora in grado di scriverne uno completo.

In sintesi, l'approccio suggerito è quello di uno sviluppo incrementale, piuttosto che "waterfall", prestando comunque attenzione alle priorità che ho esposto.

7 Consegna 7 NLD

7.1 Domande

1. Individuare altri momenti, oltre a quelli presentati (sezioni 4.4.1, 4.4.2, 4.4.3, 4.4.4) in cui è possibile usare il NLD per introdurre nuovi concetti.

2. Nel vostro percorso scolastico/universitario avete mai "sentito la necessità" di un meccanismo/costrutto di programmazione prima che vi venisse

spiegato?

7.2 Risposte

7.2.1 1 usare il NLD per introdurre nuovi concetti

Oltre agli esempi presentati nel paper [4] io userei NLD in questi casi:

- **Pattern**. Nel punto successivo racconterò della mia esperienza di NLD involontario riguardo al pattern Singleton. Trovo comunque che questo approcio sia ideale per presentare i pattern di qualsiasi tipo, soprattutto i Design Patterns.
- **Hashmap**. Mostrare che l'accesso in memoria può richiedere molto tempo, soprattutto con molti dati. Fare quindi svolgere problemi preliminari di estrazione di un dato (ad esempio usare un array o una lista) e poi fornire un dataset contenente moltissimi elementi.
- **OOP/struct**. Di nuovo, prendendo spunto dalla mia esperienza presenterei OOP tramite NLD, dato che usando strutture sempre più complesse nasce ad un certo punto la necessità di un costrutto avente una struttura decisa dallo sviluppatore. Per me ad esempio questa necessità è nata nel linguaggio C ed è stata risolta con l'utilizzo delle struct. Dalla definizione di oggetto si passa naturalmente all' object oriented programming
- **Strumenti di sincronizzazione**. Come semafori, messaggi, code, lock, mutex. Senza andare troppo nello specifico ci sono operazioni di mutua esclusione che possono essere svolte in maniera molto più intuitiva approfittando dell'astrazione. Si può certamente implementare tutti i tipi di mutua esclusione con i semafori, ma ci sono strumenti più sofisticati ed adatti alle singole esigenze. oppure si può fare l'opposto, partendo da questi strumenti più sofisticati spostarsi verso "il basso", quindi verso i semafori.

7.2.2 2 Necessità di un costrutto di programmazione

Mi è capitato due volte che io ricordi. Un esempio "voluto" si presenta nel corso "Algoritmi e Complessità", ove il professor Roversi presenta 3 problemi irrisolvibili con la nostre conoscenze attuali con lo scopo di mostrarci questa nostra "carenza". Un esempio lampante di "Necessity Learning Design", mi viene addirittura da pensare che il professore abbia sentito parlare di questa metodologia ed abbia voluto provarla. Anche il resto di quel corso presenta molti momenti di "problem solving" con interazione della classe, forse più un esempio di "Productive failure" che di "Necessity Learning Design", in linea con il discorso fatto in conclusione della conferenza riguardo al non abusare del

NDL
Invece in maniera più "naturale" mi è capitato con i Design Pattern, ricordo di aver avuto necessità di una programmazione più "strutturata". Ê successo molto tempo fa ma se non vado errato mi era capitato di aver necessità di utilizzare un Singleton (al tempo senza sapere cosa fosse) e di aver pensato "Ci sarà un modo di utilizzare una classe come se fosse un oggetto", aver cercato a lungo su internet ed aver scoperto il Singleton Pattern.

Riferimenti bibliografici

[1] Peter J. Denning. *Is Computer Science Science*. 2019. URL: https://informatica.i-learn.unito.it/pluginfile.php/338972/mod_resource/content/1/isComputerScience%20AScience.pdf.

[2] Carl Gustav Hempel. *Paradosso dei corvi*. URL: https://it.wikipedia.org/wiki/Paradosso_dei_corvi.

[3] *MANIFESTO DI VIENNA PER L'UMANESIMO DIGITALE*. 2019. URL: https://dighum.ec.tuwien.ac.at/wp-content/uploads/2019/07/Vienna_Manifesto_on_Digital_Humanism_IT.pdf.

[4] Marco SBARAGLIA Michael LODI Simone MARTINI. *A Necessity-Driven Ride on the Abstraction Rollercoaster of CS1 Programming*. URL: https://infedu.vu.lt/journal/INFEDU/article/721/info.

[5] Enrico Nardelli. *Informatica e competenze digitali: cosa insegnare?* URL: https://link-and-think.blogspot.com/2019/03/informatica-e-competenze-digitali-cosa.html.

[6] Dr. Seymour Papert. *The Gears of My Childhood*. 1980. URL: http://dailypapert.com/the-gears-of-my-childhood/.

[7] Viera K. Proulx. *Programming Patterns and Design Patterns in the Introductory Computer Science Course*. URL: http://www.ccs.northeastern.edu/home/vkp/Papers/Patterns-sigcse2000.pdf.