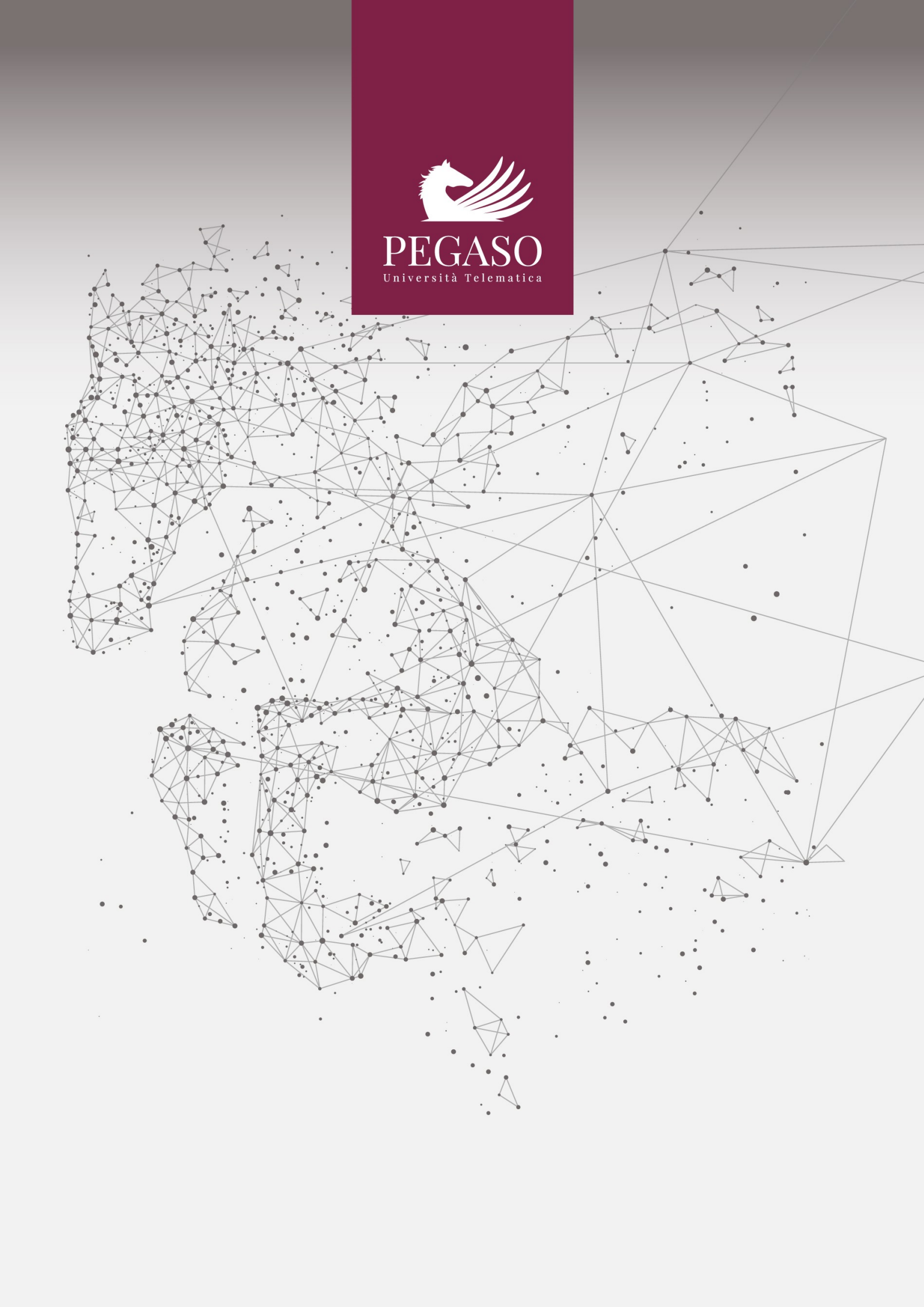
PEGASO
Università Telematica

# Indice

**1. CLASSIFICAZIONE DEI PROBLEMI ..... 3**
**2. TECNICA DI PROGETTO ..... 5**
**3. DIVIDE ET IMPERA ..... 7**
**4. LA TORRE DI HANOI ..... 9**
**BIBLIOGRAFIA ..... 13**

# 1. Classificazione dei problemi

Cos'è un problema? In ambito psicologico, possiamo parlare di problema ogni qual volta ci troviamo di fronte ad una discrepanza tra il nostro stato attuale e ciò che desideriamo: in questo caso, definire il problema rappresenta il primo passo per poterlo risolvere.

Definire il problema ci consente di riflettere maggiormente e meglio su tale evento, aiutandoci a trovare soluzioni più efficaci non dettate da meccanismi automatici o emotive.

In generale, un problema è un qualcosa che siamo chiamati a risolvere. Occorre tener presente che non tutti i problemi sono però risolvibili: il fatto di non avere una soluzione ad un problema non significa che il problema non sia interessante, anzi, in qualche caso i problemi più interessanti potrebbero anche essere quelli che non hanno ancora soluzione.

L'utilizzo di uno pseudocodice o di un flow-chart ci consente dunque di focalizzarci sulla nostra capacità di sviluppare una "intuizione algoritmica", cioè trovare possibili soluzioni a problemi, ed in particolare, soluzioni "ottimali". Analizzare un insieme di problemi ed un insieme di soluzioni buone (ed in alcuni casi ottime) sono importanti per poter costruire un vocabolario mentale per il nostro futuro come informatici.

Non ci sono regole generali per risolvere un problema in modo ottimo, tuttavia, è possibile evidenziare quattro fasi:

- Classificazione del problema.
- Caratterizzazione della soluzione.
- Tecnica di progetto.
- Utilizzo di strutture dati.

Queste fasi non sono necessariamente sequenziali.

Cerchiamo di capire in che modo è possibile classificare i problemi:

- **Problemi decisionali**
    - Sono problemi che rispondono alla domanda: il dato di ingresso soddisfa una certa proprietà?
    - Una tipica soluzione a questa classe di problemi è di tipo boolean: si / no; true / false; 0 / 1.
    - Un esempio di questo tipo di problemi è: stabilire se un grafo è connesso.
- **Problemi di ricerca**
    - Lo spazio di ricerca di questo tipo di problemi è l'insieme di "soluzioni" possibili.
    - Soluzione ammissibile: soluzione che rispetta certi vincoli.

  - Un esempio di questo tipo di problemi è: la ricerca della posizione di una sottostringa in una stringa.

- **Problemi di ottimizzazione**
  - Si ricerca una soluzione "ottima" al problema, pertanto ogni soluzione è associata ad una funzione di costo.
  - La soluzione è quella "a costo minimo".
  - Un esempio di questo tipo di problemi è: cammino più breve fra due nodi di un grafo.

La definizione formale del problema è il primo passo per poter approcciare correttamente il problema stesso: a volte la formulazione può sembrare banale ma può suggerire una prima idea di soluzione. La stessa definizione matematica può suggerire una possibile tecnica risolutiva.

# 2. Tecnica di progetto

Dopo aver introdotto il tema della classificazione di un problema, elenchiamo di seguito alcune delle tecniche più utilizzate per la progettazione di una soluzione:

- **Divide-et-impera**
  - Un problema viene suddiviso in sotto-problemi indipendenti tra loro, che vengono risolti ricorsivamente.
  - Approccio: top-down.
  - Ambito: problemi di decisione / ricerca.
- **Programmazione dinamica**
  - La soluzione viene costruita a partire da un insieme di sotto-problemi potenzialmente ripetuti.
  - Approccio: bottom-up.
  - Ambito: problemi di ottimizzazione.
- **Memoization (o annotazione)**
  - Supponiamo ad es. di avere un metodo che dobbiamo chiamare più volte e che esegue un calcolo lungo o comunque dispendioso in termini di risorse e supponiamo anche che non sappiamo prevedere con che parametri chiameremo questo metodo e quindi capiterà di chiamarlo più volte bella stessa esecuzione con gli stessi parametri e ripetere lo stesso calcolo più volte. Una strategia molto semplice è quella di utilizzare una cache per recuperare il valore se già stato calcolato.
  - Approccio: top-down (di fatto può essere indicata come la versione top-down della programmazione dinamica).
  - Ambito: problemi di ottimizzazione.
- **Tecnica greedy**
  - Approccio "ingordo": si fa sempre la scelta localmente ottima. L'ingordo compie ad ogni passo, la scelta migliore nell'immediato piuttosto che adottare una strategia a lungo termine.
  - Approccio: top-down.
  - Ambito: problemi di selezione di un sottoinsieme "ottimo" di elementi, che verificano certe proprietà.

- **Backtrack**
    - Procediamo per "tentativi", tornando ogni tanto sui nostri passi: si basa sul concetto di costruzione ed eventuale distruzione di parte della soluzione ovvero "*prova a fare qualcosa, se non funziona disfala e prova a farne un'altra*".
    - Ambito: algoritmi di visita di una struttura dati.
- **Algoritmi probabilistici**
    - Meglio scegliere con giudizio (ma in maniera costosa) o scegliere a caso ("gratuitamente"): il calcolo delle probabilità è applicato non ai dati di input, ma ai dati di output. Sono algoritmi la cui correttezza è probabilistica (Montecarlo) o sono corretti ma il cui tempo di funzionamento è probabilistico (Las Vegas).

# 3. Divide et impera

La tecnica del divide et impera prevede di dividere ricorsivamente un problema in due o più sotto-problemi di egual dimensione fin quando questi ultimi diventino di semplice risoluzione; quindi, si combinano le soluzioni al fine di ottenere la soluzione del problema dato.

La procedura della ricerca binaria (ricerca di un elemento in un array ordinato) prevede di fatto la suddivisione del problema in 2 sotto-problemi e la ricorsione solo in uno di essi e può essere ricondotta alla formulazione del divide et impera (anche se nella formulazione standard si prevede che la ricorsione avvenga per ogni dei sotto-problemi). Sebbene dunque la ricerca binaria possa essere ritenuta un "cattivo esempio" di divide et impera, ci consente di formulare un approccio assolutamente efficace soprattutto in ambito informatico e gestionale: se non tutti i sotto-problemi devono essere analizzati, è buona regola affrontare ricorsivamente i problemi più piccoli.

Tale approccio permette dunque di affrontare in modo semplice problemi anche molto difficili: la natura del divide permette inoltre di parallelizzare la computazione aumentandone l'efficienza su sistemi distribuiti o multiprocessore.

Questo tipo di approccio è tipicamente detto top down.

Un programma sviluppato secondo questa tecnica è sostanzialmente diviso in tre parti:

- **Divide**: in questa parte si procede alla suddivisione dei problemi in problemi di dimensione minore.
- **Impera**: nella seconda parte i problemi vengono risolti in modo ricorsivo; quando i sotto-problemi arrivano ad avere una dimensione sufficientemente piccola, essi vengono risolti direttamente tramite il caso base.
- **Combina**: l'ultima fase del paradigma prevede di ricombinare l'output ottenuto dalle precedenti chiamate ricorsive al fine di ottenere il risultato finale.

È importante sottolineare come non esiste una ricetta unica per tale tecnica; infatti, alcuni algoritmi che la utilizzano possono avere sostanziali differenze nelle varie fasi; a titolo esemplificativo riportiamo ad es.:

- Ordinamento Merge Sort:
    - Divide: semplice.
    - Combina: complesso.
- Ordinamento Quicksort:
    - Divide: complesso.
    - Combina: assente.

Analizziamo a titolo di esempio un'implementazione della ricerca di minimo in un array utilizzando la tecnica del divide et impera:

```
int recursive_min(int array[], int len, int start, int finish) {
  if (start == finish)
    return array[start];
  else {
    int mean = (start+finish)/2;
    return min(
      recursive_min(array,len,start,mean),
      recursive_min(array,len,mean+1,finish));
  }
}
```

***Figura 1*** *- Recursive Minimum*

Analizziamo le varie fasi della tecnica:

- Divide: si suddivide l'array in 2 parti.
- Impera: si procede ricorsivamente fino ad arrivare al caso base (start == finish).
- Combina: si combinano i risultati che si originano dalle chiamate ricorsive.

# 4. La torre di Hanoi

Analizziamo il gioco della Torre di Hanoi

***Figura 2*** *- La Torre di Hanoi*

Il gioco è costituito da:

- tre pioli (A, B, C);
- n dischi di dimensioni diverse.

Inizialmente, i dischi sono impilati in ordine decrescente nel piolo di sinistra. Lo scopo del gioco è quello di impilare in ordine decrescente i dischi sul piolo di destra senza mai impilare un disco più grande su uno più piccolo e muovendo al massimo un disco alla volta. Per fare questo è possibile utilizzare il piolo centrale come appoggio.

La proprietà matematica base è che il numero minimo di mosse necessarie per completare il gioco è:

$$2^n - 1$$

essendo n è il numero di dischi.

Ad esempio: avendo 4 dischi, il numero di mosse minime è 15.

Secondo la leggenda, i monaci di Hanoi dovrebbero effettuare almeno 18.446.744.073.709.551.615 mosse prima che il mondo finisca, essendo n=64 (supponendo che i monaci facciano una mossa al secondo il mondo finirà tra 5.845.580.504 secoli).

Osserviamo la sequenza di passi per n=3:

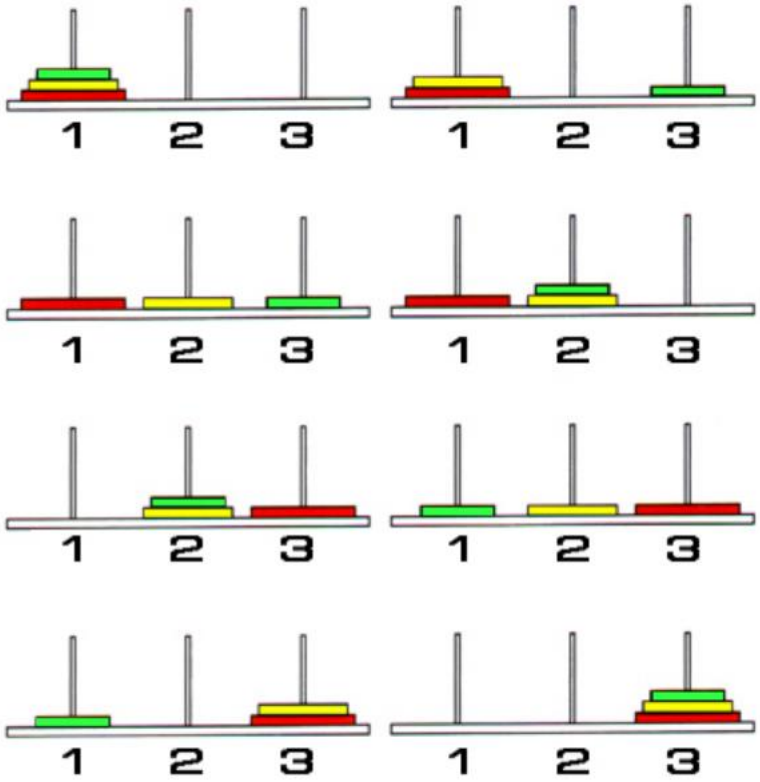


***Figura 3*** *- Torre di Hanoi con n=3*

Divide et impera e ricorsione sono alla base dell'algoritmo di soluzione della Torre di Hanoi; l'algoritmo si esprime come segue:

- Sposta i primi n-1 dischi da A a B.
- Sposta il disco n da A a C.
- Sposta n-1 dischi da B a C.

Di seguito l'implementazione in c++:

```
void hanoi(int n, int A, int C, int B) {
  if (n==1)
    cout<<A<<" -> "<<C<<endl;
  else {
    hanoi(n-1, A, B, C);
    cout<<A<<" -> "<<C<<endl;
    hanoi(n-1, B, C, A);
  }
}

int main() {
  hanoi(3,1,3,2);
}
```

***Figura 4*** *- Torre di Hanoi - c++*

La funzione hanoi viene richiamata passando i seguenti parametri:

- n=3 → numero di dischi.
- A=1 → numero del primo piolo.
- B=2 → numero del secondo piolo.
- C=3 → numero del terzo piolo.

Pertanto, in fase iniziale quello che accade è che:

- hanoi(n-1, A, B, C) → vengono spostati n-1 dischi da A a B (C è di appoggio);
- spostamento di un disco da A a C;
- hanoi(n-1, B, C, A) → vengono spostati n-1 dischi da B a C (A è di appoggio).

La sequenza di output ottenuta è la seguente:

```
> ./main
1 -> 3
1 -> 2
3 -> 2
1 -> 3
2 -> 1
2 -> 3
1 -> 3
>
```

***Figura 5*** *- Output: Torre di Hanoi*

La complessità dell'algoritmo (che è possibile dimostrare come "ottimo") è di tipo esponenziale, essendo il numero di mosse da eseguire pari a:

$$2^n - 1$$

È possibile testare la Torre di Hanoi online al seguente link: Torre di Hanoi – GeoGebra

# Bibliografia


- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione.
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione.
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley.
- Sedgewick: Algoritmi in C, Pearson, 2015.