Guida pratica per gli esercizi - Approccio Bresolin

🎯 CLASSI ESSENZIALI

📊 DEFINIZIONI RAPIDE

```
P = problemi decidibili in tempo polinomiale deterministico
NP = problemi verificabili in tempo polinomiale
NP-Hard = almeno difficili come ogni problema NP
NP-Complete = NP ∩ NP-Hard

RELAZIONE: P ⊆ NP, e P = NP ? è il problema del millennio
```

🔧 COME RICONOSCERE LA CLASSE

- **P**: Hai algoritmo polinomiale deterministico
- **NP**: Hai verificatore polinomiale (certificato facile da controllare)
- **NP-Hard**: Serve riduzione da problema NP-Hard noto

📚 PROBLEMI FONDAMENTALI DEL CORSO

🔹 PROBLEMI SU GRAFI

IndependentSet (NP-completo)

```
DEFINIZIONE: {⟨G,k⟩ | G ha insieme indipendente di k vertici}
INSIEME INDIPENDENTE: Vertici non adiacenti tra loro

VERIFICATORE: Dato insieme S, controlla |S| = k e nessun arco interno
RIDUZIONE STANDARD: 3SAT → IndependentSet (triangoli + consistenza)
```

VertexCover (NP-completo)

```
DEFINIZIONE: {⟨G,k⟩ | G ha vertex cover di k vertici}
VERTEX COVER: Ogni arco ha almeno un estremo nel set
```

```
VERIFICATORE: Dato insieme C, controlla |C| = k e ogni arco è coperto
RIDUZIONE FACILE: IndependentSet → VertexCover (complemento)
FATTO: I indipendente ⟺ V\I è vertex cover
```

Clique (NP-completo)

```
DEFINIZIONE: {⟨G,k⟩ | G ha clique di k vertici}
CLIQUE: Sottografo completo (tutti collegati tra loro)

VERIFICATORE: Dato insieme K, controlla |K| = k e tutti gli archi presenti
RIDUZIONE FACILE: IndependentSet → Clique (grafo complemento)
```

3-Coloring (NP-completo)

```
DEFINIZIONE: {⟨G⟩ | G è 3-colorabile}
3-COLORABILE: Vertici colorabili con 3 colori, adiacenti ≠ colore

VERIFICATORE: Data colorazione, controlla validità
RIDUZIONE COMPLESSA: 3SAT → 3-Color (gadget variabili + clausole)
NOTA: 2-Color è in P (test bipartiteness)
```

HamiltonianPath (NP-completo)

```
DEFINIZIONE: {⟨G,s,t⟩ | G ha cammino hamiltoniano da s a t}
HAMILTONIANO: Cammino che visita ogni vertice esattamente una volta

VERIFICATORE: Dato cammino, controlla che sia valido e hamiltoniano
USO: Per problemi che richiedono "ordinamento" di oggetti
```

🔹 PROBLEMI SAT

3SAT (NP-completo BASE)

```
DEFINIZIONE: {⟨ϕ⟩ | ϕ è formula 3CNF soddisfacibile}
3CNF: ϕ = (l₁∨l₂∨l₃) ∧ (l₄∨l₅∨l₆) ∧ ... (3 letterali per clausola)

VERIFICATORE: Dato assegnamento, controlla se soddisfa ϕ
IMPORTANZA: Problema più usato come sorgente per riduzioni
Cook-LEVIN: Primo problema dimostrato NP-completo
```

2SAT (in P)

```
DEFINIZIONE: {⟨ϕ⟩ | ϕ è formula 2CNF soddisfacibile}

ALGORITMO P: Grafo delle implicazioni + SCC
DISTINGUE: 2SAT facile, 3SAT difficile
```

🔹 PROBLEMI NUMERICI

SubsetSum (NP-completo)

```
DEFINIZIONE: {⟨S,t⟩ | ∃S' ⊆ S tale che sum(S') = t}
INPUT: Insieme numeri S, target t

VERIFICATORE: Dato sottoinsieme S', controlla se somma a t
RIDUZIONE DA: 3SAT (codifica clausole come equazioni numeriche)
ALGORITMO DP: O(nt) - pseudo-polinomiale
```

SetPartitioning (NP-completo)

```
DEFINIZIONE: {⟨S⟩ | S partizionabile in due parti di uguale somma}

VERIFICATORE: Data partizione, controlla uguaglianza somme
RELAZIONE: Caso speciale di SubsetSum con t = sum(S)/2
RIDUZIONI: SubsetSum ⇔ SetPartitioning (bidirezionali)
```

🧩 RIDUZIONI STANDARD PER ESAMI

🔹 DA 3SAT (le più comuni)

3SAT → IndependentSet

```
COSTRUZIONE:
• Per ogni clausola (a ∨ b ∨ c): crea triangolo con vertici a,b,c
• Per ogni coppia xᵢ, ¬xᵢ: collega con arco (consistenza)
• k = numero clausole

IDEA: Insieme indipendente sceglie 1 letterale vero per clausola
```

3SAT → 3-Color

```
COSTRUZIONE COMPLESSA:
• Triangolo base con colori TRUE, FALSE, BASE
• Gadget per variabile: forza xᵢ = TRUE o FALSE
• Gadget per clausola: forza ≥1 letterale TRUE
NOTA: Costruzione molto tecnica, studia bene gli esempi
```

3SAT → SubsetSum

```
COSTRUZIONE:
• Per ogni variabile: due numeri (TRUE/FALSE)
• Per ogni clausola: numeri "slack" per bilanciare
• Codifica in base k+1 dove k = max(clausole, variabili)
IDEA: Soluzione codifica assegnamento soddisfacente
```

🔹 RIDUZIONI FACILI

IndependentSet ↔ VertexCover

```
RIDUZIONE: ⟨G,k⟩ → ⟨G, |V|-k⟩
FATTO MATEMATICO: I indipendente ⟺ V\I vertex cover
```

IndependentSet ↔ Clique

```
RIDUZIONE: ⟨G,k⟩ → ⟨Ḡ,k⟩ (grafo complemento)
FATTO: I indipendente in G ⟺ I clique in Ḡ
```

SubsetSum ↔ SetPartitioning

```
SubsetSum → Partition: ⟨S,t⟩ → ⟨S ∪ {sum(S)-t, t}⟩
Partition → SubsetSum: ⟨S⟩ → ⟨S, sum(S)/2⟩
```

🎯 STRATEGIA PER ESERCIZI

🔧 PER DIMOSTRARE NP

```
SCHEMA FISSO:
1. Definisci certificato (es: sottoinsieme, assegnamento, cammino)
2. Scrivi verificatore polinomiale
```

```
3. Analizza complessità (deve essere polinomiale)
4. Dimostra completezza + soundness
```

🔧 PER DIMOSTRARE NP-HARD

```
SCELTA PROBLEMA SORGENTE:
• Problema generale → 3SAT
• Problema su grafi (selezione) → IndependentSet
• Problema su grafi (copertura) → VertexCover
• Problema numerico → SubsetSum o SetPartitioning
• Problema con ordinamento → HamiltonianPath

SCHEMA FISSO:
1. Scegli sorgente appropriato
2. Costruisci riduzione f
3. Dimostra f polinomiale
4. Dimostra correttezza (⇒) e (⇐)
```

🔧 RICONOSCIMENTO PATTERN COMUNI

Quando usare 3SAT come sorgente:

- Problema ha natura "soddisfacimento vincoli"
- Servono scelte binarie per oggetti
- Problema generale senza struttura particolare

Quando usare IndependentSet:

- Problema chiede "sottoinsieme che evita conflitti"
- Oggetti non possono essere scelti insieme
- Vincoli di incompatibilità

Quando usare VertexCover:

- Problema chiede "piccolo insieme che copre tutto"
- Vincoli devono essere "controllati" o "coperti"
- Ottimizzazione di risorse limitate

Quando usare HamiltonianPath:

- Problema richiede ordinamento/sequenza di oggetti
- Ogni oggetto usato esattamente una volta
- Problemi di scheduling o percorsi

📝 TEMPLATE VELOCI PER ESAMI

🔧 Template NP

```
TEOREMA: [Problema] ∈ NP

DIMOSTRAZIONE:
Certificato: [descrizione]
Verificatore V: "Su input ⟨istanza, certificato⟩:
1. [controlli formato]
2. [controlli proprietà]
3. Se tutto ok, ACCETTA"

Complessità: O([analisi])
Correttezza: [argomento breve]
```

🔧 Template NP-Hard

```
TEOREMA: [Problema] è NP-hard

DIMOSTRAZIONE: [Sorgente] ≤ₚ [Problema]

Riduzione f: "Su input [istanza_sorgente]:
1. [costruzione istanza target]
2. Restituisci [istanza_target]"

Correttezza:
• (⇒): [se sorgente SI, allora target SI]
• (⇐): [se target SI, allora sorgente SI]
Polinomialità: [analisi tempo costruzione]
```

✅ CHECKLIST ESSENZIALE

Problemi che DEVI saper dimostrare NP:

- [ ] IndependentSet (certificato = sottoinsieme)
- [ ] VertexCover (certificato = sottoinsieme)
- [ ] 3SAT (certificato = assegnamento)

- [ ] SubsetSum (certificato = sottoinsieme)
- [ ] HamiltonianPath (certificato = sequenza vertici)

Riduzioni che DEVI saper fare:

- [ ] 3SAT → IndependentSet (triangoli + consistenza)
- [ ] IndependentSet → VertexCover (complemento)
- [ ] IndependentSet → Clique (grafo complemento)
- [ ] SubsetSum ↔ SetPartitioning (bidirezionale)

Fatti che devi ricordare:

- [ ] 2SAT ∈ P, 3SAT NP-completo
- [ ] I indipendente ⟺ V\I vertex cover
- [ ] I indipendente in G ⟺ I clique in Ḡ
- [ ] 3SAT è il problema "universale" per riduzioni

Strategie da ricordare:

- [ ] Certificato deve essere verificabile in tempo polinomiale
- [ ] Riduzione deve preservare soluzioni in entrambe le direzioni
- [ ] Scegli problema sorgente in base alla struttura del target
- [ ] Sempre dimostrare (⇒) E (⇐) per correttezza riduzione