GUIDA COMPLETA AGLI ESERCIZI NP E NP-HARD

Metodologia sistematica per ogni tipo di problema

📋 INDICE RAPIDO

PARTE 1: DIMOSTRARE CHE UN PROBLEMA È NP {#parte-1}

🎯 OBIETTIVO: Costruire un verificatore polinomiale

📝 SCHEMA STANDARD

Step 1: Identificare il certificato

- **Domanda chiave**: "Quale informazione aggiuntiva mi permetterebbe di verificare velocemente la soluzione?"
- **Esempi tipici**:
  - Per IndependentSet: l'insieme S di vertici candidato
  - Per VertexCover: l'insieme C di vertici candidato
  - Per SAT: un assegnamento di verità alle variabili
  - Per Hamiltoniano: la sequenza di vertici del cammino
- **Vincolo dimensione**: |certificato| ≤ polinomio in |input|

Step 2: Definire formalmente il verificatore

- **Template**: V = "Su input ⟨istanza, certificato⟩:"
- **Esempio**: V = "Su input ⟨⟨G,k⟩, S⟩:"

Step 3: Descrivere l'algoritmo di verifica

- **Controlli da fare** (in questo ordine):
  1. **Controllo formato**: certificato ha formato corretto?
  2. **Controllo dimensione**: certificato rispetta i vincoli dimensionali?
  3. **Controllo proprietà**: certificato soddisfa le proprietà richieste?
- **Output**: ACCETTA se tutti i controlli passano, RIFIUTA altrimenti

Step 4: Analisi della complessità

- **Per ogni controllo**, calcola il tempo richiesto
- **Somma totale** deve essere polinomiale in |input|
- **Scrivi esplicitamente**: "Tempo totale: O(...))"

Step 5: Prova di correttezza

- **Completezza**: Se istanza ∈ problema, allora ∃ certificato che fa accettare V
- **Soundness**: Se V accetta ⟨istanza, certificato⟩, allora istanza ∈ problema

🔧 TEMPLATE VERIFICATORE

```
VERIFICATORE V:
Input: ⟨istanza_problema, certificato⟩

1. [CONTROLLO FORMATO]
   • Verifica che certificato abbia il formato atteso
   • Se no, RIFIUTA

2. [CONTROLLO DIMENSIONE]
   • Verifica vincoli dimensionali del certificato
   • Se no, RIFIUTA

3. [CONTROLLO PROPRIETÀ]
   • Verifica che certificato soddisfi le proprietà del problema
   • Se no, RIFIUTA

4. ACCETTA

COMPLESSITÀ: O(...)
CORRETTEZZA: [dimostrazione completezza + soundness]
```

PARTE 2: DIMOSTRARE CHE UN PROBLEMA È NP-HARD {#parte-2}

🎯 OBIETTIVO: Mostrare una riduzione polinomiale da un problema NP-Hard noto

📝 SCHEMA STANDARD RIDUZIONE

Step 1: Scelta del problema sorgente

- **Problemi di riferimento** (dal più usato al meno usato):
  1. **3SAT** (formula booleana in forma normale congiuntiva con 3 letterali per clausola)
  2. **IndependentSet** (insieme di vertici non adiacenti)
  3. **VertexCover** (insieme di vertici che tocca ogni arco)
  4. **CircuitSAT** (soddisfacibilità di circuiti booleani)
  5. **SAT** (soddisfacibilità generale)

Step 2: Definizione della funzione di riduzione

- **Template**: f = "Su input istanza_A:"
- **Costruzione**: Trasforma l'istanza di A in un'istanza di B
- **Output**: Restituisce istanza_B

Step 3: Descrizione dettagliata della trasformazione

- **Ogni componente** dell'istanza A deve essere mappato sistematicamente in B
- **Mantenere traccia** delle corrispondenze per la prova di correttezza
- **Usare costruzioni modulari** (per ogni clausola, per ogni vertice, etc.)

Step 4: Prova di correttezza (FONDAMENTALE)

- **Dimostrazione bidirezionale**: **Direzione (⇒)**:
  - Se istanza_A ∈ A, allora f(istanza_A) ∈ B
  - "Supponiamo che istanza_A abbia la proprietà..."
  - "Allora possiamo costruire una soluzione per f(istanza_A)..."

  Direzione (⇐):
  - Se f(istanza_A) ∈ B, allora istanza_A ∈ A
  - "Supponiamo che f(istanza_A) abbia una soluzione..."
  - "Allora possiamo costruire una soluzione per istanza_A..."

Step 5: Analisi complessità della riduzione

- **Tempo di costruzione**: deve essere polinomiale
- **Dimensione output**: deve essere polinomiale rispetto all'input

🔧 TEMPLATE RIDUZIONE

```
RIDUZIONE A ≤p B:

FUNZIONE f:
Input: istanza_A
1. [COSTRUZIONE COMPONENTI]
   • Per ogni elemento x in istanza_A:
     - Crea elemento corrispondente y in istanza_B
     - Mantieni relazione x ⇔ y

2. [ASSEMBLAGGIO]
   • Combina tutti i componenti in istanza_B valida

3. Output: istanza_B

CORRETTEZZA:
- Direzione (⇒): istanza_A ∈ A ⟹ f(istanza_A) ∈ B
  Dimostrazione: [...]

- Direzione (⇐): f(istanza_A) ∈ B ⟹ istanza_A ∈ A
  Dimostrazione: [...]

COMPLESSITÀ: O(...)
```

PARTE 3: DIMOSTRARE CHE UN PROBLEMA È NP-COMPLETO {#parte-3}

🎯 OBIETTIVO: Combinare le due dimostrazioni precedenti

📝 SCHEMA COMPLETO

Teorema: Problema X è NP-Completo

Dimostrazione:

Parte 1: X ∈ NP

- [Segui PARTE 1 - Verificatore polinomiale]

Parte 2: X è NP-Hard

- [Segui PARTE 2 - Riduzione da problema NP-Hard noto]

Conclusione: Poiché X ∈ NP e X è NP-Hard, allora X è NP-Completo. □

PARTE 4: ESEMPI PRATICI STEP-BY-STEP {#parte-4}

📚 ESEMPIO 1: IndependentSet ∈ NP

Problema: IndependentSet = {⟨G,k⟩ | G ha un insieme indipendente di dimensione k}

Soluzione:

Step 1 - Certificato: Un sottoinsieme S ⊆ V di vertici

Step 2 - Verificatore:

```
V = "Su input ⟨⟨G,k⟩, S⟩:
1. Controlla che S ⊆ V(G)
2. Controlla che |S| = k
3. Per ogni coppia u,v ∈ S:
   • Se (u,v) ∈ E(G), RIFIUTA
4. ACCETTA"
```

Step 3 - Complessità: $O(|S|^2) = O(|V|^2)$ = polinomiale

Step 4 - Correttezza:

- Se G ha indip.set di dim k, allora certificato = quell'insieme fa accettare V
- Se V accetta ⟨⟨G,k⟩, S⟩, allora S è indip.set di dim k in G

📚 ESEMPIO 2: VertexCover è NP-Hard

Riduzione: IndependentSet ≤p VertexCover

Soluzione:

Step 1 - Funzione di riduzione:

```
f = "Su input ⟨G,k⟩:
1. Costruisci G' = G (stesso grafo)
2. Poni k' = |V| - k
3. Output: ⟨G', k'⟩"
```

Step 2 - Correttezza:

Direzione (⇒): Se G ha indip.set I di dim k

- Allora C = V \ I è vertex cover di dim |V| - k
- Perché: ogni arco (u,v) ha almeno un endpoint in C (non possono essere entrambi in I)

Direzione (⇐): Se G ha vertex cover C di dim |V| - k

- Allora I = V \ C è indip.set di dim k
- Perché: nessun arco può avere entrambi gli endpoint in I (sennò C non coprirebbe quell'arco)

Step 3 - Complessità: O(1) - banalmente polinomiale

📚 ESEMPIO 3: 3SAT ≤p IndependentSet

Costruzione del grafo (TECNICA FONDAMENTALE):

Step 1 - Creazione vertici:

- Per ogni clausola ($l_1 \lor l_2 \lor l_3$): crea 3 vertici, uno per letterale
- Etichetta: ogni vertice rappresenta un letterale

Step 2 - Archi di clausola:

- In ogni "triangolo" di clausola: collega tutti i 3 letterali tra loro
- **Effetto**: al massimo 1 letterale per clausola può essere nell'insieme indipendente

Step 3 - Archi di consistenza:

- Collega ogni letterale x con ogni occorrenza di ¬x
- **Effetto**: non si possono scegliere contemporaneamente x e ¬x

Step 4 - Parametro:

- k = numero di clausole

Correttezza:

- **Se φ soddisfacibile**: scegli 1 letterale vero per clausola → indip.set di dim k
- **Se G ha indip.set di dim k**: deve contenere 1 vertice per clausola, tutti consistenti → assegnamento che soddisfa φ

PARTE 5: PROBLEMI DI RIFERIMENTO E RIDUZIONI STANDARD {#parte-5}

🗂️ GERARCHIA DELLE RIDUZIONI

Problemi Base (usa questi come sorgente):

1. **CircuitSAT** → SAT → 3SAT
2. **3SAT** → {IndependentSet, VertexCover, Clique}
3. **IndependentSet** ↔ VertexCover ↔ Clique (riduzioni facili)

Riduzioni Standard:

3SAT → IndependentSet:

- **Tecnica**: Triangoli per clausole + archi di consistenza
- **Difficoltà**: Media
- **Quando usare**: Per problemi sui grafi

IndependentSet → VertexCover:

- **Tecnica**: Complemento (I indip. ⟺ V\I vertex cover)
- **Difficoltà**: Facile
- **Quando usare**: Sempre quando possibile

VertexCover → SetCover:

- **Tecnica**: Ogni arco = insieme, ogni vertice = elemento
- **Difficoltà**: Facile
- **Quando usare**: Per problemi di copertura

3SAT → 3-Coloring:

- **Tecnica**: Gadget per variabili + gadget per clausole
- **Difficoltà**: Alta
- **Quando usare**: Per problemi di colorazione

🎨 TECNICHE DI COSTRUZIONE AVANZATE

Gadget Modulari:

- **Per variabili**: Strutture che forzano scelte binarie
- **Per clausole**: Strutture che verificano soddisfacimento
- **Per vincoli**: Connessioni che propagano vincoli

Tecniche di Amplificazione:

- **Duplicazione**: Ripeti componenti per aumentare parametri
- **Catene**: Collega componenti in sequenza per forzare ordini
- **Cross-prodotti**: Combina gadget per vincoli multipli

🚫 ERRORI COMUNI DA EVITARE

Nel Verificatore:

- ❌ Dimenticare controlli di formato/dimensione
- ❌ Algoritmo di verifica non polinomiale
- ❌ Non dimostrare completezza E soundness

Nelle Riduzioni:

- ❌ Dimenticare una delle due direzioni della correttezza
- ❌ Riduzione non polinomiale
- ❌ Mapping non ben definito tra soluzioni

Negli Scritti:

- ❌ Non definire formalmente il problema target
- ❌ Saltare passaggi "ovvi" nella dimostrazione
- ❌ Non specificare la complessità esplicitamente

🎯 CHECKLIST FINALE

Per ogni esercizio NP:

- [ ] Certificato identificato e dimensione polinomiale
- [ ] Verificatore definito formalmente
- [ ] Tutti i controlli implementati correttamente
- [ ] Complessità calcolata e polinomiale
- [ ] Completezza e soundness dimostrate

Per ogni esercizio NP-Hard:

- [ ] Problema sorgente NP-Hard scelto appropriatamente
- [ ] Funzione di riduzione definita costruttivamente
- [ ] Entrambe le direzioni della correttezza dimostrate
- [ ] Complessità della riduzione polinomiale
- [ ] Mapping tra soluzioni esplicito

Per ogni esercizio NP-Completo:

- [ ] Parte NP completata correttamente
- [ ] Parte NP-Hard completata correttamente
- [ ] Conclusione esplicita di NP-completezza

💡 SUGGERIMENTI STRATEGICI

Scelta del Problema Sorgente:

- **3SAT**: Per problemi logici o combinatori generali
- **IndependentSet**: Per problemi sui grafi
- **VertexCover**: Quando IndependentSet è troppo complesso
- **CircuitSAT**: Solo se necessario (solitamente evita)

Organizzazione della Dimostrazione:

1. **Enunciato chiaro** del teorema
2. **Struttura a sezioni** (NP + NP-Hard)
3. **Prove di correttezza complete**
4. **Conclusioni esplicite**

Debugging delle Riduzioni:

- **Testa su esempi piccoli** prima di scrivere la dimostrazione generale
- **Verifica che il mapping preservi soluzioni** in entrambe le direzioni
- **Controlla casi limite** (grafi vuoti, formule senza clausole, etc.)