Progetto: Chinese Remainder Theorem Solver con Generazione LaTeX Automatica

Obiettivo

Creare un sistema Python che risolve sistemi di congruenze lineari usando il Teorema Cinese del Resto e genera automaticamente documentazione LaTeX completa con tutti i passaggi matematici dettagliati, basandosi sulla struttura estratta dall'esercizio di esempio fornito.

Struttura del Progetto

```
chinese_remainder_solver/
├── main.py                    # Interface principale
├── core/
│   ├── crt_solver.py          # Algoritmi matematici core
│   ├── euclidean.py           # Algoritmo di Euclide esteso
│   └── math_utils.py          # Utilità matematiche (MCD, MCM, fattorizzazione)
├── latex/
│   ├── generator.py           # Generatore LaTeX dinamico
│   └── templates/
│       ├── base.tex           # Template documento base
│       ├── compatibility.tex  # Template verifica compatibilità
│       ├── euclidean.tex      # Template algoritmo Euclide
│       ├── solution.tex       # Template calcolo soluzione
│       └── verification.tex   # Template verifiche aggiuntive
├── output/                    # File generati
└── examples/                  # Esempi di test
```

Algoritmo Core (da crt_solver.py)

Input

Sistema di congruenze nella forma:

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$

Algoritmo (seguendo la struttura dell'esempio PDF)

PASSO 1: Verifica Compatibilità

- Calcola $d = MCD(m_1, m_2)$
- Verifica se $d \mid (a_2 - a_1)$
- Se NO: sistema incompatibile, termina
- Se SÌ: procedi

PASSO 2: Algoritmo di Euclide Esteso

- Applica algoritmo di Euclide a (m_1, m_2)
- Registra TUTTI i passaggi per il LaTeX
- Usa sostituzione "a ritroso" per trovare coefficienti di Bézout
- Trova u, v tali che: $d = u \cdot m_1 + v \cdot m_2$

PASSO 3: Soluzione Particolare

- Calcola: $(a_2 - a_1) = k \cdot d$ per qualche k
- Trova soluzione particolare: $x_0 = a_1 + k \cdot u \cdot m_1$
- Verifica sostituendo nel sistema

PASSO 4: Insieme Soluzioni

- Calcola $MCM(m_1, m_2) = (m_1 \cdot m_2)/MCD(m_1, m_2)$
- Soluzione generale: $S = \{x_0 + MCM \cdot t : t \in \mathbb{Z}\}$
- Normalizza x_0 nell'intervallo [0, MCM)

Struttura Dati Output

python

```python
class CRTSolution:
    # Input
    congruences: List[Tuple[int, int]]  # [(a₁,m₁), (a₂,m₂)]

    # Calcoli intermedi
    gcd_value: int
    gcd_steps: List[str]               # Passi algoritmo Euclide
    bezout_coeffs: Tuple[int, int]     # Coefficienti u, v
    compatibility_check: bool

    # Risultato
    particular_solution: int           # x₀
    period: int                        # MCM
    solution_set: str                  # Rappresentazione matematica

    # Verifiche extra (opzionali)
    divisibility_checks: List[Tuple[int, bool]]  # [(divisore, risultato)]
```

Generatore LaTeX (latex/generator.py)

Template System

Ogni template LaTeX usa placeholder nella forma `{{variable_name}}`:

base.tex:

latex

```latex
\documentclass[12pt]{article}
\usepackage[utf8]{inputenc}
\usepackage{amsmath, amssymb}
\begin{document}
\title{{{title}}}
\date{{{date}}}
\maketitle

{{content}}

\end{document}
```

compatibility.tex:

latex

```latex
\textbf{Passo 1: Compatibilità.}
Grazie al teorema cinese del resto, il sistema è compatibile se e solo se
$$\gcd({{m1}}, {{m2}}) \mid {{a2}} - {{a1}} = {{diff}}$$

Decomponendo in fattori primi:
$${{m1}} = {{m1_factorization}}, \quad {{m2}} = {{m2_factorization}}$$

Pertanto $\gcd({{m1}}, {{m2}}) = {{gcd_value}}$.

{{#if compatible}}
La condizione è verificata: ${{gcd_value}} \mid {{diff}}$. Il sistema è compatibile.
{{else}}
La condizione NON è verificata: ${{gcd_value}} \nmid {{diff}}$. Il sistema è incompatib
{{/if}}
```

Logica di Generazione

python

```python
class LaTeXGenerator:
    def __init__(self, solution: CRTSolution):
        self.solution = solution
        self.templates = self.load_templates()

    def generate_compatibility_section(self) -> str:
        # Popola template compatibilità

    def generate_euclidean_section(self) -> str:
        # Genera tabella passi Euclide + sostituzione a ritroso

    def generate_solution_section(self) -> str:
        # Calcolo soluzione particolare e generale

    def generate_verification_section(self) -> str:
        # Verifiche aggiuntive se richieste

    def compile_full_document(self) -> str:
        # Assembla documento completo
```

Interface Utente (main.py)

Modalità Interactive

python

```python
def main():
    print("=== Chinese Remainder Theorem Solver ===")

    # Input sistema
    system = input_congruence_system()

    # Opzioni aggiuntive
    options = {
        'check_divisibility': input("Verificare divisibilità per un numero? (y/n): "),
        'show_factorizations': input("Mostrare fattorizzazioni? (y/n): "),
        'output_format': input("Formato output (pdf/tex/both): ")
    }

    # Risoluzione
    solution = solve_crt_system(system, options)

    # Generazione LaTeX
    latex_doc = generate_latex_documentation(solution, options)

    # Output
    save_and_compile(latex_doc, options['output_format'])
```

Modalità Batch

python

```python
# Supporto per file JSON con multiple test cases
def batch_mode(input_file: str):
    test_cases = load_test_cases(input_file)
    for case in test_cases:
        solution = solve_crt_system(case['system'], case['options'])
        generate_output(solution, case['output_path'])
```

Funzionalità Avanzate

1. Gestione Errori

- Sistema incompatibile → Genera LaTeX con spiegazione del perché
- Input invalidi → Messaggi di errore chiari
- Overflow numerici → Gestione con warning

2. Ottimizzazioni

- Caching risultati MCD/MCM per input simili
- Rappresentazione efficiente numeri grandi
- Parallelizzazione per batch processing

3. Estensibilità

- Supporto sistemi n×n (iterando l'algoritmo 2×2)
- Plugin system per verifiche custom
- Export in formati multipli (LaTeX, HTML, Markdown)

Casi di Test Richiesti

Test Case 1 (dall'esempio PDF)

```
Input: x ≡ 28 (mod 108), x ≡ 64 (mod 78)
Expected: x ≡ 1000 (mod 1404), divisibile per 4
```

Test Case 2 (incompatibile)

```
Input: x ≡ 1 (mod 6), x ≡ 3 (mod 9)
Expected: Sistema incompatibile (MCD(6,9)=3 non divide 3-1=2)
```

Test Case 3 (caso semplice)

```
Input: x ≡ 2 (mod 3), x ≡ 3 (mod 5)
Expected: x ≡ 8 (mod 15)
```

Requisiti Tecnici

- Python 3.8+
- Librerie: `math`, `typing`, `pathlib`, `json`
- LaTeX compiler (pdflatex) per generazione PDF
- Template engine custom (no dipendenze esterne)

Output Atteso

Per ogni input, il sistema deve produrre:

1. **solution.tex**: Documento LaTeX completo con tutti i passaggi
2. **solution.pdf**: PDF compilato (se richiesto)
3. **solution_data.json**: Dati computazionali per debugging
4. **Console output**: Riassunto risultati

Il LaTeX generato deve essere identico nella struttura e completezza all'esempio fornito nel PDF, ma con valori dinamici basati sull'input specifico.