ATIVIDADE DE REFATORAÇÃO

Curso: Engenharia de Software
Matéria: Algoritmos avançados
Professor: Glauco Vinicius Scheffel

Equipe: Alexandre Tessaro Vieira, Edson Borges Polucena, Leonardo Pereira Borges, Richard Schmitz Riedo e Wuelliton Chistian dos Santos

Católica SC
Centro Universitário

INTRODUÇÃO

- Escolhemos a equipe do Lucas para refatorar o código.
- Essa atividade teve como foco principal melhorar a qualidade do código por meio de boas práticas de engenharia de software, com base nas técnicas de refatoração do Martin Fowler, Refactoring Guru e Catálogo de Refatorações.
- Nosso objetivo foi aumentar a legibilidade, modularidade e testabilidade do código, além de preparar a base para futuras melhorias e facilitar a manutenção.

O QUE FOI REFATORADO:

1. RENOMEAÇÃO DE VARIÁVEIS E FUNÇÕES

- Antes: Nomes pouco descritivos como trial, limit, scout(), entre outros.
- Depois: Nomes mais expressivos como trial_counters, limit_trials, generate_scout_bees().
- Motivo: Aplicamos a refatoração Rename Variable / Rename Method para melhorar a clareza e autoexplicatividade do código.

O QUE FOI REFATORADO:

2. EXTRAÇÃO DE FUNÇÕES

- Criamos funções como calculate_fitness(), select_best_sources() e generate_neighbor().
- Motivo: Usamos a técnica Extract Method para quebrar funções longas e isolar responsabilidades.

2. EXTRAÇÃO DE FUNÇÕES

- Cada função tem uma responsabilidade única (Single Responsibility Principle).
- Código ficou mais modular, testável e reutilizável.
- Clareza melhorada: ao ler o loop principal, você entende o que está acontecendo sem entrar nos detalhes internos.

Antes:

```
for i in range(n):
    total_value = 0
    total_weight = 0
    for j in range(len(items)):
        if solution[i][j] == 1:
            total_value += items[j][0]
            total_weight += items[j][1]
    if total_weight <= max_weight:
        fitness[i] = total_value
    else:
        fitness[i] = 0

best_sources = []
for i in range(n):
    if fitness[i] > threshold:
        best_sources.append(solution[i])

# Em seguida, começa a gerar vizinhos...
```

Depois:

```
def calculate_fitness(solution, items, max_weight):
    total_value, total_weight = 0, 0
    for j in range(len(items)):
        if solution[j] == 1:
            total_value += items[j][0]
            total_weight += items[j][1]
    return total_value if total_weight <= max_weight else 0

def select_best_sources(solutions, fitnesses, threshold):
    return [sol for sol, fit in zip(solutions, fitnesses) if fit > threshold]

def generate_neighbor(solution):
    neighbor = solution[:]
    index = random.randint(0, len(solution) - 1)
    neighbor[index] = 1 - neighbor[index]
    return neighbor
```

O QUE FOI REFATORADO:

3. ISOLAMENTO DE RESPONSABILIDADES

- Criamos uma classe Bee (ou estrutura equivalente) para encapsular informações como posição e aptidão.
- Separação entre lógica do algoritmo e leitura de dados.
- Motivo: Aplicamos Move Method e Encapsulate Variable para separar preocupações, tornando o código mais orientado a objetos e modular.

3. ISOLAMENTO DE RESPONSABILIDADES

- A classe Bee encapsula estado e comportamento → melhora a coerência e permite evoluir o código com mais segurança.
- O código ficou mais orientado a objetos.
- Separação de preocupações (dados x lógica) facilita testes, manutenção e futuras mudanças.

Antes:

```
solutions = []
fitnesses = []
for i in range(num_bees):
    solution = [random.randint(0, 1) for _ in range(len(items))]
    total_value, total_weight = 0, 0
    for j in range(len(solution)):
        if solution[j] == 1:
            total_value += items[j][0]
            total_weight += items[j][1]
    if total_weight <= max_weight:
        fitness = total_value
    else:
        fitness = 0
    solutions.append(solution)
    fitnesses.append(fitness)
```

Depois:

```
class Bee:
    def __init__(self, solution, items, max_weight):
        self.solution = solution
        self.fitness = self.calculate_fitness(items, max_weight)

    def calculate_fitness(self, items, max_weight):
        total_value, total_weight = 0, 0
        for i in range(len(self.solution)):
            if self.solution[i] == 1:
                total_value += items[i][0]
                total_weight += items[i][1]
        return total_value if total_weight <= max_weight else 0
```

CatólicaSC
Centro Universitário

O QUE FOI REFATORADO:

4. PARÂMETROS E CONFIGURAÇÕES

- Centralizamos as configurações em um dicionário ou classe de parâmetros.
- Motivo: Facilitar a reutilização e alteração de parâmetros, reduzindo o acoplamento.

O QUE FOI REFATORADO:

5. TESTES AUTOMATIZADOS COM PYTEST

- Adicionamos testes para funções como calculate_fitness(), is_valid_solution(), etc.
- Motivo: Garantir que as refatorações não alterassem o comportamento original. Seguindo TDD pós-refatoração.

O QUE FOI REFATORADO:

6. DOCUMENTAÇÃO

- Criamos arquivos README.md, refatoracao.md e processo.md.
- Incluímos: como rodar o projeto, decisões tomadas, técnicas aplicadas e plano de refatoração.
- Motivo: Melhorar a comunicação técnica e permitir reuso por outros desenvolvedores.

O QUE FOI REFATORADO:

7. ANÁLISE DE QUALIDADE

- Utilizamos ferramentas como pylint e flake8.
- Corrigimos más práticas, code smells e aumentamos a cobertura de testes.
- Motivo: Garantir aderência a padrões de qualidade e reduzir riscos de manutenção futura.

RESULTADOS DA REFATORAÇÃO:

- Código mais limpo e organizado
- Redução de duplicação e acoplamento
- Aumento da legibilidade e manutenibilidade
- Possibilidade de reuso e extensão facilitada
- Base sólida para testes e evolução do projeto

OBRIGADO

CatólicaSC
Centro Universitário