

Relatório — Projeto classe_pilha

Nome: Bruno Videira Pinho DRE: 119.161.539

Estruturas de Dados Utilizadas

O projeto utiliza a estrutura de dados pilha (stack), implementada na classe **Pilha** no arquivo **pilha.py**. A pilha é baseada no módulo **array** do Python, permitindo o armazenamento de dados inteiros ('i') ou caracteres unicode ('u'). Exceções são usadas para tratar erros como: pilha cheia, pilha vazia, troca inválida e tipo incorreto de dado.

Divisão de Módulos

- **pilha.py**: Implementa a classe **Pilha** e exceções relacionadas. Fornece métodos para empilhar, desempilhar, verificar se está cheia/vazia, trocar elementos do topo e obter o tamanho.
- **fill_region.py**: Implementa o algoritmo de flood fill (preenchimento por inundação) de duas maneiras diferentes uma usa a classe **Pilha** + loops e a outra utiliza recursão.
- **input.txt**: Arquivo de entrada para o algoritmo de flood fill, representando uma matriz binária com um ponto inicial marcado por 'X'.
- **torre_hanoi.py**: Implementa a solução do problema da Torre de Hanoi utilizando três instâncias da classe **Pilha** para representar os pinos. Inclui métodos para resolver o problema, mostrar o estado atual e contar movimentos.
- **relatorio.md**: Este relatório.

Principais Rotinas e Funções

- **Pilha**
 - **empilha(data)**: Adiciona um elemento ao topo da pilha.
 - **desempilha()**: Remove e retorna o elemento do topo.
 - **pilha_esta_vazia()** / **pilha_esta_cheia()**: Verificam o estado da pilha.
 - **troca()**: Troca de posição os dois elementos do topo.
 - **tamanho()**: Retorna o número de elementos na pilha.
- **fill_region.py**
 - **flood_fill_recursive**: Implementação do flood fill por recursão.
 - **flood_fill_loop**: Implementação do flood fill usando a classe **Pilha** e loops.
 - **show_matrix**: Exibe a matriz formatada no terminal.
- **torre_hanoi.py**
 - **TorreHanoi**: Classe principal para resolver o problema.
 - **solve()**: Resolve a Torre de Hanoi.
 - **show_state()**: Exibe o estado atual dos pinos.
 - **show_moves_count()**: Exibe o número de movimentos realizados (executado apenas no final de **solve()**).

Complexidade de Tempo e Espaço

- **Pilha:** Todas as operações principais (**empilha**, **desempilha**, **troca**) são $O(1)$.
- **Flood Fill:** Complexidade $O(N*M)$ para matrizes $N \times M$, tanto na versão recursiva quanto iterativa.
- **Torre de Hanoi:** Complexidade $O(2^n)$, onde n é o número de discos (pelos resultados que tive brincando com esse algoritmo o número de passos é sempre $2^n - 1$).

Problemas e Observações

- O uso do módulo **array** garante eficiência, mas limita os tipos de dados aceitos.
- O flood fill pode causar RecursionError para matrizes muito grandes na versão recursiva (pelo seu bem não tente rodar ele com o arquivo **big_matrix.txt**).
- O código da Torre de Hanoi é flexível para diferentes números de discos, mas a visualização pode ficar difícil para valores muito altos. Dependendo das capacidades do seu terminal não seria possível ver todos os passos para resolução da torre.

Conclusão

O projeto demonstra o uso prático da estrutura de dados pilha (stack) em dois algoritmos clássicos: flood fill e torre de hanoi. Além disso, a separação das funcionalidades em diferentes módulos facilita futuras expansões e adaptações. Este projeto cumpre seu objetivo, de apresentar como uma pilha pode ser utilizada para resolver problemas recorrentes em análise de dados.