Classe Pilha - Preenchimento de região

Estruturas de Dados Utilizadas

A base do projeto é a classe “Pilha”, que segue o princípio LIFO (Last In, First Out). Internamente, a Pilha utiliza um array.array do Python para armazenamento, garantindo eficiência e a restrição de tipos homogêneos.

A matriz de caracteres, que representa a área a ser preenchida, é armazenada como uma lista de listas em Python, onde cada sublista representa uma linha da matriz.

Divisão de Módulos e Organização

Para manter a clareza e facilitar a manutenção, o código foi dividido logicamente:

- **Definição de Exceções:** Classes de exceção personalizadas (PilhaCheiaErro, PilhaVaziaErro, TipoErro) foram definidas no início para lidar com condições específicas de erro da Pilha.
- **Classe Pilha:** A implementação da estrutura de dados Pilha foi encapsulada em uma classe dedicada.
- **Funções Auxiliares:** Funções como ler_matriz_de_arquivo e imprimir_matriz foram criadas para lidar com as operações de I/O (entrada e saída) e visualização, respectivamente.
- **Função Principal do Algoritmo (preencher_regiao_com_pilha):** O coração da solução, contendo a lógica do Flood Fill. Ela interage diretamente com a classe Pilha.
- **Bloco if __name__ == "__main__"::** Este bloco contém a lógica de execução principal do programa, gerenciando a leitura do arquivo, a interação com o usuário para os passos de visualização e a chamada da função de preenchimento.

Descrição das Rotinas e Funções

- **Exceções (PilhaCheiaErro, PilhaVaziaErro, TipoErro):** São classes simples que herdam de Exception, usadas para sinalizar condições de erro específicas durante as operações da pilha.

- **Pilha.__init__(self, capacidade_maxima, tipo_dado='u'):** Construtor da Pilha. Inicializa a capacidade, o array interno (self.elementos) e o ponteiro topo. Garante que a capacidade seja um inteiro positivo.
- **Pilha.empilha(self, dado):** Adiciona um dado ao topo da pilha. Realiza verificações de **tipo de dado** (conforme especificação) e de **capacidade** (lançando TipoErro ou PilhaCheiaErro se necessário).
- **Pilha.desempilha(self):** Remove e retorna o dado do topo da pilha. Lança PilhaVaziaErro se a pilha estiver vazia.
- **Pilha.pilha_esta_vazia(self):** Retorna True se a pilha não contiver elementos.
- **Pilha.pilha_esta_cheia(self):** Retorna True se a pilha atingiu sua capacidade máxima.
- **Pilha.tamanho(self):** Retorna o número de elementos atualmente na pilha.
- **Pilha.troca(self):** Troca os dois elementos do topo da pilha. Requer no mínimo dois elementos para operar.
- **ler_matriz_de_arquivo(nome_arquivo):** Lê uma matriz de caracteres de um arquivo de texto, linha por linha, e retorna-a como uma lista de listas.
- **imprimir_matriz(matriz, passo_atual=None):** Exibe a matriz no terminal, limpando a tela para uma visualização dinâmica. Troca '1's por espaços e '0's por '#' para um melhor efeito visual.
- **preencher_regiao_com_pilha(matriz, linha_inicial, coluna_inicial, passos_entre_apresentacao):** Implementa o algoritmo de Flood Fill. Inicia empilhando a posição de partida. Em um loop, desempilha uma posição, a "preenche" (muda para '0'), e então empilha seus vizinhos adjacentes que ainda são '1's e estão dentro dos limites da matriz.

Complexidades de Tempo e Espaço

- Pilha: As operações da Pilha têm complexidade de tempo O(1). O espaço ocupado é O(C), onde C é a capacidade máxima.
- Preenchimento de Região: O algoritmo de Flood Fill possui complexidade de tempo O(M * N) no pior caso, pois cada célula da matriz (M linhas, N colunas) pode ser visitada e processada uma única vez. A complexidade de espaço, que se refere ao tamanho máximo da Pilha em uso, também é O(M * N) no pior cenário, onde toda a matriz forma uma única região a ser preenchida.

Problemas e Observações Encontrados Durante o Desenvolvimento

- **Capacidade da Pilha:** A Pilha foi implementada com uma capacidade máxima fixa. Para o problema de preenchimento, a capacidade foi definida como num_linhas * num_colunas, garantindo que há espaço suficiente para todas as células da matriz no pior caso.

- **Pilha Transbordando:** O número de elementos na pilha cresce indefinidamente, levando a PilhaCheiaErro, ou Stack Overflow se fosse recursivo. Pois não foram marcar as células já processadas, ou já adicionadas à pilha, como "visitadas", o que também provocou um looping infinito.

- **Verificação de Limites:** As coordenadas dos vizinhos não estavão dentro dos limites da matriz (0 <= nr < num_linhas e 0 <= nc < num_colunas). Isso resultou em IndexError ao tentar acessar posições inválidas.

Conclusão

Este projeto demonstra a importância da escolha de estruturas de dados adequadas para resolver problemas de forma eficiente e como a pilha pode gerenciar o fluxo de execução e evitar potenciais problemas de estouro de pilha (Stack Overflow) em conjuntos de dados maiores.

Alunos:

Davidson Diógenes Vasconcelos da Silva Santos	DRE: 123531495
Mônica de Sousa Amaral	DRE: 119160444
Naya da Silva Nascimento	DRE: 119160428