UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA UFRJ

GABRIEL SANTA BARBARA RODRIGUES SOUZA - 123356007

JULIA FERNANDA TERRA SOUZA - 123460458

SAMUEL DO NASCIMENTO CAROLA - 123537899

Classe Pilha

Rio de Janeiro

2024

Classe Pilha

1.Introdução

O desenvolvimento deste projeto focou na implementação do algoritmo Flood Fill utilizando duas abordagens distintas: recursiva e iterativa com pilha. A estrutura de dados central foi uma pilha genérica, implementada através de uma lista em Python, que armazena coordenadas (x, y) para a versão iterativa. A pilha foi projetada com operações básicas como empilhar, desempilhar e verificar estado, além de incluir exceções personalizadas para casos de overflow e underflow. A matriz, representada como uma lista de listas de strings, permitiu acesso eficiente aos elementos e fácil manipulação durante o preenchimento. A organização do código foi modular, separando claramente a lógica da pilha, as funções de leitura e exibição da matriz, e os próprios algoritmos de Flood Fill, o que facilitou a manutenção e o teste individual de cada componente.

2. Arquitetura do sistema

As rotinas principais incluíram o flood_recursivo, que utiliza chamadas recursivas para explorar vizinhos, marcando posições como preenchidas, e o flood_pilha, que substitui a recursão por uma pilha explícita, processando cada coordenada de forma iterativa. A função mostrar_matriz foi essencial para visualizar o progresso, convertendo ‘1’ em espaços vazios e ‘0’ em caracteres sólidos para melhor distinção. A complexidade de tempo para ambos os métodos foi O(nm), onde n e m são as dimensões da matriz, pois cada célula é visitada no pior caso. No entanto, a versão recursiva possui complexidade de espaço O(nm) devido à pilha de chamadas, enquanto a iterativa também é O(n*m) em espaço, mas evita estouro de pilha em matrizes grandes.

3.Problemas e Observações Durante o Desenvolvimento

Durante o desenvolvimento, alguns desafios surgiram, como o risco de estouro de pilha na abordagem recursiva e o consumo excessivo de memória na versão iterativa devido ao empilhamento de todos os vizinhos antes do processamento. Além disso, a falta de validação para matrizes irregulares ou entradas inválidas, como múltiplos ‘X’, poderia levar a comportamentos inesperados. A comparação final entre os resultados mostrou que ambos os métodos produziram saídas idênticas, validando a corretude das implementações. A versão com pilha destacou-se como a mais robusta para cenários práticos, especialmente em matrizes extensas, enquanto a recursiva serviu como uma demonstração clara do algoritmo, porém com limitações.

4.Resultados e conclusão

Conclui-se que o projeto atingiu seus objetivos, oferecendo uma comparação eficaz entre as duas abordagens e destacando as vantagens da implementação iterativa em termos de estabilidade. A modularidade do código e o uso de tipagem estática aumentaram a legibilidade e a segurança, enquanto a visualização passo a passo auxiliou no entendimento do processo. Para futuras melhorias, sugere-se a adição de validações de entrada mais rigorosas e a exploração de otimizações, como o uso de filas para um gerenciamento mais eficiente de memória.