PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

GABRIELA HOFFMANN ROXO

RELATÓRIO TRABALHO 2

Algoritmo e Estrutura de Dados I

PORTO ALEGRE

2024

SUMÁRIO

INTRODUÇÃO

1.1 Descrição do trabalho

A proposta do trabalho é implementar uma calculadora de expressões matemáticas com base na estrutura de dados pilha. Para organizar o código optei por criar as seguintes classes além das disponibilizadas: Pilha.java e Calculadora.java.

A base para processar as equações provém da classe Pilha, cujos métodos essenciais são: push(e), pop(), top(), isEmpty(), size() e clear(). O desenvolvimento em estrutura encadeada permite a manipulação eficiente e flexível dos elementos, tornando-a adequada para armazenar e processar os elementos da conta.

Diante disso, deve-se utilizar o arquivo "LeituraArqJava.java" para processar as expressões contidas no arquivo "expressoes3.txt". Essa abordagem permite a conexão entre as classes do arquivo.

Cada expressão é avaliada individualmente, e o resultado é apresentado juntamente com o tamanho máximo da pilha. Para isso, a classe Calculadora consta com estes métodos para auxiliar: isOperator(), isOpen(), isClose(), prioridadeOperador(), processarOperador(), verificarErros(), calcularEquacao() e processExpressoes(). Caso ocorram erros de sintaxe, como chaves, parênteses ou colchetes mal formados, erros de operandos insuficientes para realizar o cálculo ou erro de carácter inválido, a calculadora os identifica e os apresenta de forma clara.

1.2 Contextualização da solução feita

No desenvolvimento da calculadora de expressões matemáticas, a classe Pilha é utilizada para armazenar temporariamente os elementos das expressões enquanto são processadas, além de manipular os elementos, controlar o tamanho e limpar a pilha. Durante o cálculo de uma expressão matemática, os números e operadores são adicionados à pilha conforme a ordem de prioridade das operações. Isso permite que a calculadora avalie corretamente a expressão, garantindo que as operações sejam realizadas na ordem correta.

A solução das expressões é coordenada e processada pela classe Calculadora, composta por duas pilhas para armazenar operadores e valores (instâncias da classe Pilha),

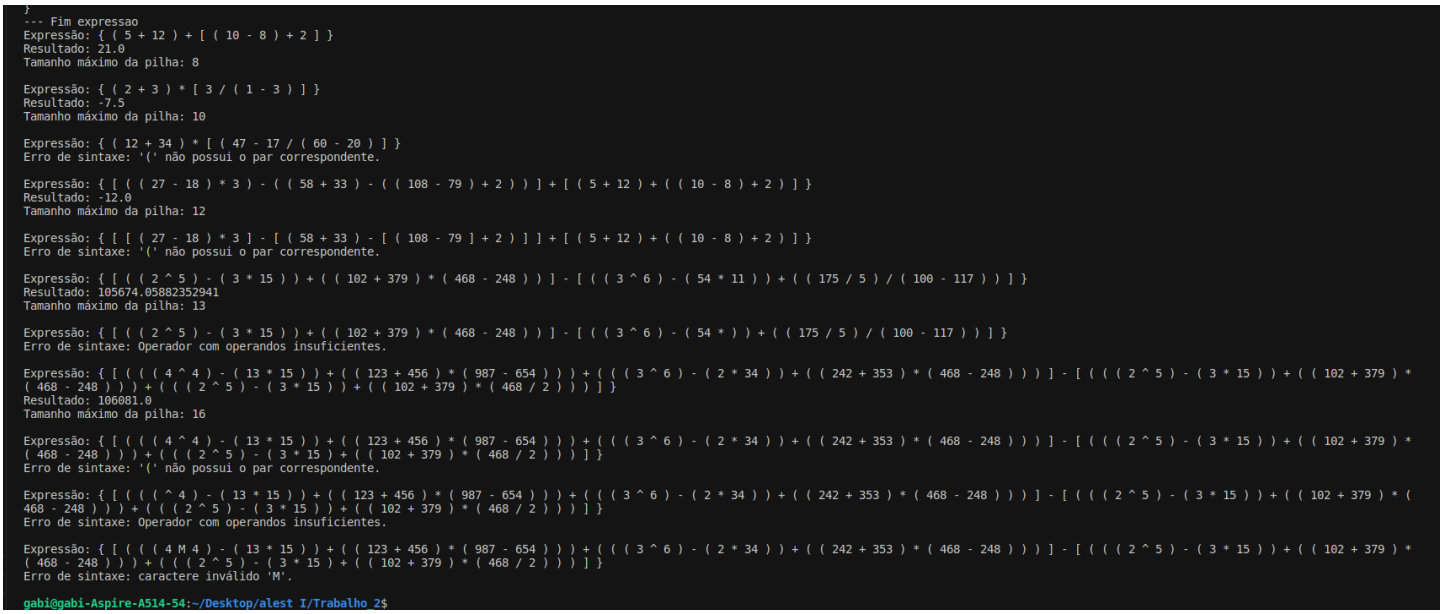
uma variável para indicar se houve erro durante o processo e outra variável para rastrear o tamanho máximo da pilha. Sua estrutura inicia pelos métodos de verificação - se um caractere é um operador matemático (adição, subtração, divisão, potência, multiplicação) e se um caractere representa a abertura ou fechamento de um parêntese, colchete ou chave. Em sequência, estipula-se a prioridade entre os operadores com base na ordem de precedência, ou seja, quanto maior o valor retornado, maior a prioridade. Por fim, o método verificarErros() conta o número de colchetes, parênteses e chaves de abertura e fechamento, garantindo que estejam balanceados, caso contrário emite uma mensagem de erro sintático.

Para realizar os cálculos, o método processarOperador() efetua a operação entre os dois operandos do topo da pilha de valores, utilizando o operador especificado. O resultado é empilhado de volta na pilha de valores. Já o método calcularEquacao() processa uma única expressão, dividindo-a em tokens - números, operadores e símbolos - empilhando os valores na pilha de valores e os operadores na pilha de operadores. Ele usa as regras de precedência dos operadores e símbolos para calcular o resultado da expressão.

A resposta é emitida pelo método processExpressoes(), que recebe uma lista de expressões matemáticas, chamando os métodos calcularEquacao() e verificarErros() para cada uma delas. Optei por utilizar ArrayList, pois fornece uma estrutura flexível e eficiente para armazenar e manipular os elementos. Além disso, criei uma instância de classe Calculadora na classe LeituraArqJava para processar as expressões contidas na lista.

DESCRIÇÃO ALGORITMO

2.1 Resultados obtidos (print da saída)



Expressões Mal Formadas:

- Nas questões 3 e 9, há um número ímpar de parênteses, indicando que uma abertura não possui seu par de fechamento.
- Na equação 5, o erro está relacionado à tentativa de estipular um par entre um parêntese e um colchete, o que é uma configuração inválida na sintaxe matemática.
- As expressões 7 e 10 estão tentando realizar operações de multiplicação e potenciação com apenas um operando.
- O erro na última expressão ocorre devido ao caractere entre dois números, impossibilitando a realização de uma operação matemática válida.

Link para o vídeo com a explicação do algoritmo :

https://youtu.be/FYQpJhIm12g

CONCLUSÃO

3.1 Dificuldades enfrentadas

Inicialmente, havia um problema em rastrear o tamanho máximo da pilha adequadamente. Isso envolveu a identificação dos pontos corretos no código onde o tamanho máximo deveria ser atualizado, especialmente durante as operações de empilhamento e desempilhamento. Um desafio significativo foi garantir que a expressão matemática fosse válida em termos de sintaxe. Isso incluiu a verificação de parênteses, colchetes e chaves correspondentes, bem como a validação de caracteres inválidos e a presença de operandos suficientes para cada operador.

Além disso, o código envolve o uso de duas pilhas: uma para os valores numéricos e outra para os operadores. Coordenar corretamente as operações entre essas duas pilhas, especialmente durante a avaliação da expressão matemática, foi uma dificuldade que exigiu atenção especial. Outro desafio foi identificar e corrigir bugs no código existente, especialmente quando os resultados não estavam de acordo com o esperado. Isso exigiu uma análise detalhada do código e uma compreensão do funcionamento das pilhas e das operações matemáticas.

3.2 Complexidade O

A complexidade depende principalmente do método calcularEquacao() da classe Calculadora, pois realiza o processamento principal das expressões aritméticas. Para dividir a equação em tokens, o método split() é linear em relação ao tamanho da string a ser dividida. Após isso, a iteração sobre cada token segue linear. Para cada token, o método pode empilhar valores na ‘valorPilha’ e operadores na ‘operadorPilha’. A complexidade do empilhamento é constante, pois as pilhas são implementadas com estruturas encadeadas, e a inserção em uma pilha encadeada é uma operação de tempo constante. Dessa forma, em uma expressão matemática bem formada, o número de de operadores geralmente é proporcional ao número de valores na expressão. Portanto, a complexidade total do código também é linear em relação ao número de expressões da lista.