Algoritmos e Estrutura de Dados I

Lorran Marques
Escola Politécnica - PUCRS

September 22, 2022

Abstract

Neste relatório está descrito a implementação de uma calculador para números complexos utilizando conceitos de uma pilha encadeada. O acesso à pilha é feito exclusivamente através dos métodos de inserção (push()), remoção (pop()) e verificação se está vazia (isempty()). Ao final, executamos a implementação para oito casos de testes e computamos o tamanho máximo atingido pela pilha duranta a execução, e o tamanho da pilha ao final da execução.

1 Método

1.1 Implementação da Pilha

A pilha foi implementada como uma pilha encadeada de tuplas de duas posições. Tuplas são imutáveis, dessa forma impedimos quaisquer alterações não desejadas nos valores da Pilha. As tuplas foram pensadas de forma que o valor na primeira posição represente a parte real do número, e o valor na segunda posição represente a parte imaginária.

1.2 Implementação das Operações

As operações básicas de soma, subtração e multiplicação foram implementadas da seguinte forma:
Seja $z_1 = (a + bi)$ e $z_2 = (c + di)$

$$z_1 + z_2 = ([a + c] + [b + d]) \tag{1}$$

$$z_1 * z_2 = ([a * c] + [b * d]) \tag{2}$$

A operação de divisão é definida matematicamente como a multiplicação do numerador e denominador pelo conjugado do denominador, generalizando esse conceito ficamos com:

$$\frac{z_1}{z_2} = \frac{(a + bi)}{(c + bi)} \times \frac{(c - bi)}{(c - bi)} = \frac{ac + bd + (cb - ad)i}{c^2 + d^2} \tag{3}$$

Para a operação de divisão foi realizado uma verificação para que, em casos de divisão por zero, uma mensagem seja exibida ao usuário e os valores de denominador e numerador sejam devolvidos à pilha.

Essas operações retiram os dois últimos dados da pilha, opera-os e insere o resultado na pilha.

A operação de mundança de sinal ("chs") foi implementada removendo a última tupla da pilha, multiplicando ambas posições por -1 e devolvendo o resultado à pilha.

O conjugado ("conj") é definido matematicamente como:

$$\overline{z} = (a - bi) \tag{4}$$

Dessa forma, retiramos a última tupla da pilha e multiplicamos apenas o valor da segunda posição por -1. O resultado devolvemos à pilha.

O inverso de um número complexo ("inv"), representado matematicamente como z_{-1} é definido como:

$$z_{-1} = \frac{1}{z} = \frac{1}{a + bi} = \frac{a - bi}{a^2 + b^2} \tag{5}$$

Retiramos o último valor da pilha, calculamos o inverso utilizando a equação acima e devolvemos o resultado à pilha.

Assim como na operação de divisão, aplicamos uma validação para evitar tentativas de divisão por 0.

O valor absoluto de um número complexo "abs" é a distância do número de zero. Podemos calculá-lo a partir do teorema de pitagoras e teremos ao fim:

$$|z| = \sqrt{a^2 + b^2} \tag{6}$$

Como o resultado dessa operação será sempre um número real positivo, optamos por devolver o resultado à pilha como o um número complexo com parte imaginária = 0. Ou seja, $|z| = (\sqrt{a^2 + b^2} + 0i)$. E a tupla será armazenada como (|z|,0).

A operação de troca "swap" remove os dois últimos valores da pilha e os devolve na ordem inversa em que foram retirados. E a operação de de duplicar "dup" remove o último valor da pilha e o insere novamente duas vezes.

1.3 Software

O software foi escrito em **Python** com auxílio da biblioteca **NumPy**.

O algoritmo inicia criando uma instância da classe Pilha e lê o arquivo de texto com as operações e números e armazena-os em uma lista de strings.

Uma função chamada operações(Pilha p, String operacao) foi criada para lidar com as operações.

Para cada linha da entrada, tentamos converter em dois números inteiros, caso resulte em erro, assumimos que seja uma operação e chamamos a função operações(), conforme trecho abaixo:

```
for linha in linhas:
        try:
            c,d = linha.split(" ")
            tuple = (int(c),int(d))
            pilha.push(tuple)
        except:
            if linha == "quit":
                break
            operacoes(pilha,linha)
```

Para sabermos o tamanho máximo atingido pela pilha e o tamanho da pilha ao final da execução, foi criada uma variável tamanho_pilha que é incrementada a cada operação de push(), e decrementada a cada operação de pop(). Junto ao incremento/decremento, é feito uma verificação se é o maior valor assumido pela variável, e, em caso positivo, armazena-o na variável max_tam_pilha (inicializada em 0). Uma forma mais eficiente seria adicionar

essas operações à classe pilha, entretanto, como o acesso à pilha deveria ser feito exclusivamente pelos métodos push(), pop() e isEmpty(), isso não seria possível.

A fim de agilizar a execução, foi criada uma outra lista, utilizando a compreensão de listas do Python, com as entradas de todos os oito casos entregues para execução.

O código fonte será entregue junto deste relatório, também poderá ser acessado através deste repositório do GitHub

2 Resultados

Executando os casos de testes encontramos os seguintes valores para Tamanho Máximo da Pilha e Tamanho da Pilha ao Final do Programa:

Caso	Tamanho Máximo	Tamanho ao Final
Caso 1	28	11
Caso 2	36	19
Caso 3	43	17
Caso 4	69	11
Caso 5	87	43
Caso 6	133	133
Caso 7	97	17
Caso 8	81	35