UNIVERSIDADE FEDERAL DE MINAS GERAIS
ALGORITMOS E ESTRUTURA DE DADOS III

Trabalho Prático 0 – Notação Polonesa Reversa

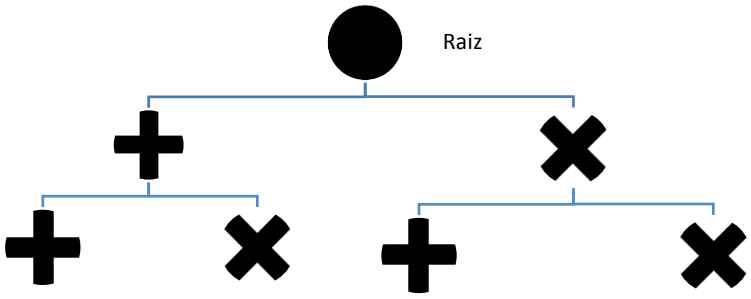
Deiziane Natani da Silva – TD1

1. Introdução

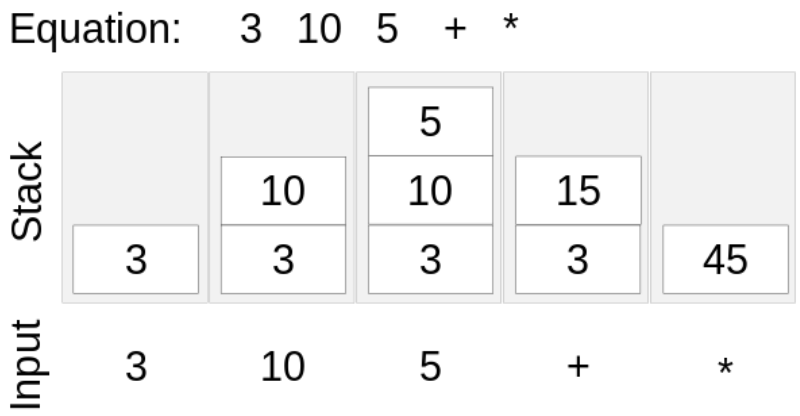
O trabalho tem como objetivo praticar os conceitos de Algoritmos e Estrutura de Dados 2. O TP consiste em resolver o problema de João, que é descobrir os sinais aritméticos representados por "?" que estão faltando numa expressão. É necessário descobrir todas as possíveis combinações e apresenta-las como saída.

2. Solução do problema

Para solucionar o problema e ajudar João, ao receber uma entrada, são verificados quantos operadores existem nessa expressão. A cada operador encontrado, este é adicionado a uma árvore binaria. Essa arvore foi criada com a intenção de obter todas as possíveis combinações de sinais da expressão. Como regra de implementação, foi decidido que todos os ramos à direita serão sinais de multiplicação, e a esquerda serão sinais de adição. Dessa forma, a árvore geraria todas as permutações de símbolos possíveis, como mostra a figura abaixo:



Note que a raiz sempre será nula, e terá apenas apontadores para a esquerda e direita. Em seguida, a arvore é percorrida em pré-ordem, ou seja, esquerda-raiz-direita. Dessa forma, a ordem dos operadores é respeitada como pedido na especificação do trabalho. Também é criada uma pilha para realizar a operação sobre os números. Sempre ao percorrer a arvore até um nó folha, uma função é chamada e realiza as operações sobre os números na pilha. Por exemplo, na primeira execução desta função os operadores da árvore serão todas os "+". Na segunda vez, os operadores serão "++*", e assim por diante. Essa função realiza todas as operações utilizando a sequência de operadores extraídos da arvore, e verifica se ao final o resultado é igual ao resultado esperado. Caso seja, esta imprime a sequência de operadores. A pilha funciona da seguinte maneira:



3. **Análise de Complexidade**

   Pilha – a inserção na pilha (push) é sempre O(1), assim como a remoção (pop). A complexidade de espaço da pilha é O(n), onde n é a quantidade de operandos da expressão.

   Árvore – Na inserção e na extração dos operadores, todos os nós são visitados em pré ordem. O número máximo de chamadas recursivas à função pré-ordem é igual à altura da árvore. Ou seja, o procedimento é O(h), com h sendo a altura da arvore. Uma árvore binária completa tem 2^h– 1 nós, onde h é a altura da árvore. Logo, numa árvore binária completa o número de chamadas recursivas à função pré-ordem é O(h) = O(log(N)), onde N é o número de nós da árvore. Cada vez que a arvore alcança uma folha, uma pilha é criada. Como as chamadas são recursivas, elas têm maior custo de espaço, pois há maior consumo de memória. A complexidade de espaço das duas funções, inserção e extração de símbolos, é O(h) pois é preciso armazenar uma chamada de função recursiva na memória para cada nó à medida que se desce recursivamente na árvore, e nunca se pode ir mais fundo do que O (h) na árvore (antes de retornar podemos nos livrar dos nós já totalmente processados).

4. **Avaliação Experimental**

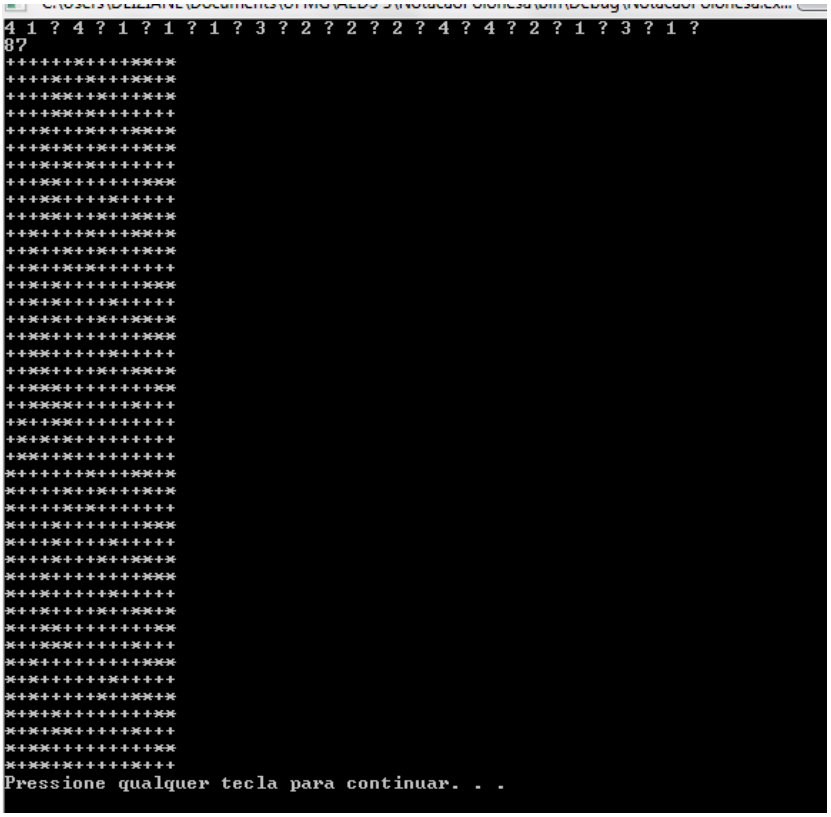
   Foram feitos testes utilizando os casos de testes toy disponibilizados no moodle. O algoritmo se mostrou eficaz em todos os casos, resultando no output correto.

   Também foram feitos testes em outros casos disponibilizados por colegas no fórum.

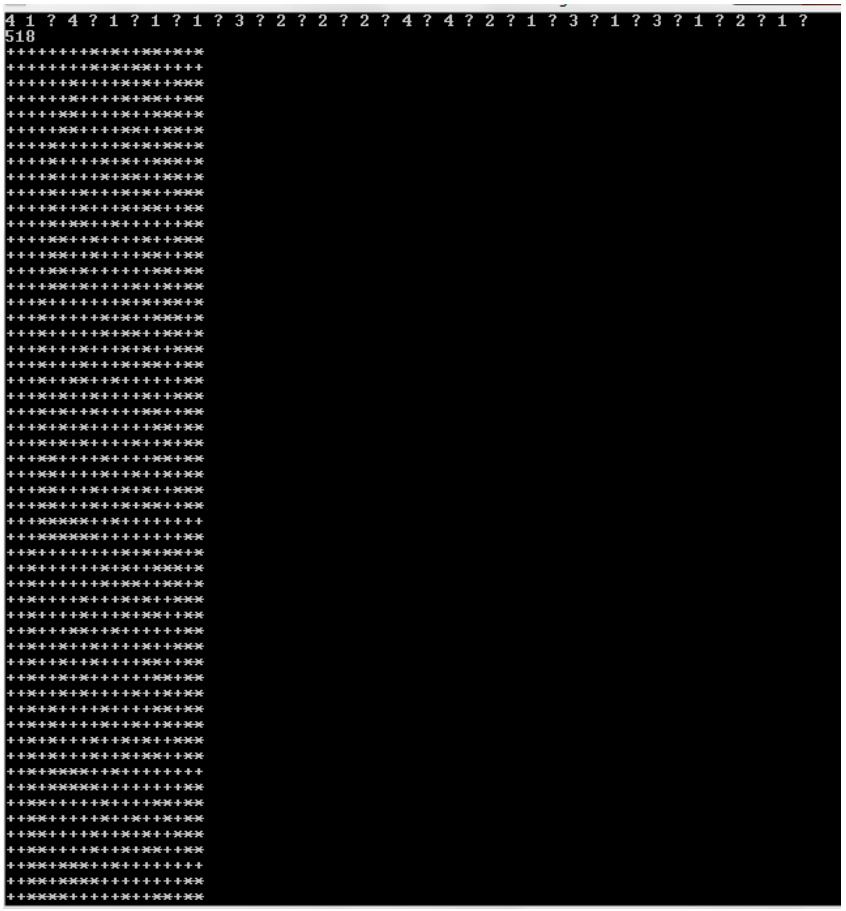
```
4 1 ? 
4
4 1 ? 4 ?
16
4 1 ? 4 ? 1 ?
9
4 1 ? 4 ? 1 ? 1 ?
10
4 1 ? 4 ? 1 ? 1 ? 1 ?
9
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ?
27
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ?
54
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ?
31
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ?
33
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ?
37
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ?
41
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ?
43
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ?
82
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ?
86
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ?
87
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ?
258

4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ?
258
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ?
261
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ?
518
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ?
1036
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ? 3 ?
3108
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ? 3 ? 3 ?
1042
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ? 3 ? 3 ? 1 ?
1043
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ? 3 ? 3 ? 1 ? 2 ?
1045
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ? 3 ? 3 ? 1 ? 2 ? 1 ?
1046
4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ? 3 ? 3 ? 1 ? 2 ? 1 ? 2 ?
2090
```

Os casos pequenos (parecidos com os toy) executaram perfeitamente. Execução do caso da linha 13:



Outros casos maiores – com grande número de operadores e operandos também funcionaram. Exemplo da expressão da linha 38:



Porém, como a arvore de operadores cresce exponencialmente a cada novo operador adicionado a expressão, a próxima expressão da linha 39, por exemplo, que tem apenas um operador a mais, não é executada até o final.

```
37  4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ?
38  518
39  4 1 ? 4 ? 1 ? 1 ? 1 ? 3 ? 2 ? 2 ? 2 ? 4 ? 4 ? 2 ? 1 ? 3 ? 1 ? 3 ? 1 ? 2 ? 1 ? 2 ?
40  1036
```

1- Expressões da linha 37 e 39. Note que há apenas um operador e operando a mais na expressão 39.

5. **Referências**

Casos de testes disponibilizados no GitHub. Disponível em:

<https://gist.github.com/Macmod/16f6d9bc7fff1cfb4ecc7f8ab51898c3>. Acesso em: 20 abr 2017.