Árvore de expressão

Renato Borges Boaventura[1]

[1] Instituto Federal de Minas Gerais - Formiga.MG (IFMG)

RESUMO

Este artigo tem como propósito mostrar em linhas gerais de como o código fora implementado e quais as escolhas feitas durante sua implementação. Serão mostradas cada TAD usadas no programa, e em que situações cada um delas foram usadas, para a manipulação das expressões passadas ao programa.

PALAVRA CHAVE:

AED; ALGORÍTMOS; PROGRAMAÇÃO; TAD.

INTRODUÇÃO

No início desse artigo, mostrará de forma breve de como fora feito o trabalho. Será mostrado as TADs utilizadas, as operações disponíveis, como o programa deve ser executado via linha de comando, a ordem de complexidade geral do programa e quais as principais escolhas feitas na construção do algoritmo. O programa foi escrito utilizando o editor de texto *Sublime Text*, e todos os testes foram realizados no sistema operacional *linux*.

DESENVOLVIMENTO

Foram usadas as seguintes TADs na construção do programas:

Árvore Binária e pilha de Árvore

É inviável descrever cada uma das TADs separadamente, já que elas foram totalmente dependentes na construção da árvore, que será usada em todo o programa. A árvore fora construída utilizando uma *pilha de árvore*, em que cada nodo seu corresponde a um operando ou a um operador. Quando o operando é lido pelo programa, um nó é criado e uma posição da pilha é apontada para o nó que possui o operando. Assim que um operador for lido, é retirado os dois nós consecutivos do topo da pilha, que serão linkados pelo operador.

Pilha de ponto flutuante(aritmética)

Após a árvore binária ser construída, as operações serão feitas com o auxílio da pilha aritmética. Será feito o encaminhamento pós ordem, em que cada operando encontrado será empilhado. Assim que um operador for encontrado, será desempilhado os dois últimos elementos, e será feito a operação entre os dois de acordo com o operador encontrado.

Expressão

Essa TAD não é essencial, visto que ela foi criada para melhor organização do programa, ela poderia ser escrita no *main*. Seu objetivo é aplicar as operações e manipular as funções das demais TADs.

Para executar o programa será necessário passar via linha de comando a forma que o arquivo será lido. Há o *in1*, que lerá os dados de entrada via teclado, nessa opção será possível passar apenas uma expressão por execução. Há, também, o *in2*, que lerá as expressões de um arquivo texto, cujo nome terá que ser informado via linha de comando, logo após a escolha do tipo de entrada que será escolhido.

Haverá, também, dois tipos de saída a ser escolhida pelo usuário. A *out1*, que mostrará os tipos de encaminhamento e o resultado final na tela, e o *out2*, que precede o nome do arquivo de saída. Exemplo : *nome_do_programa nome_do_tipo_de_entrada *nome_do_arquivo_de_entrada nome_do_tipo_de_saída *nome_do_arquivo_de_saída*.

Foram feitos inúmeros testes com inúmeras expressões, nos quais os resultados e os tipos de notações obtidos foram satisfatórios. Foi usado como critério, a notação polonesa (notação de prefixo) e a notação polonesa inversa (notação pós-fixada), nos quais as saídas obtidas foram satisfeitas com tal critério.

A expressão $5+(1+2)*4-3$ fora usada como base para os testes. As notações ficaram da seguinte forma: pós-fixada: 5 1 2 + 4 * + 3 −; prefixa: $-\ +\ 5\ *\ +\ 1\ 2\ 4\ 3$; infixa obtida sem tratamento dos parênteses: $(((5.000000)+(((1.000000)+(2.000000))*(4.000000)))-(3.000000))$. O resultado obtido fora 14.

Em linhas gerais, a complexidade do algoritmo deverá ser $O(n)$. Tal complexidade é obtida através de uma análise geral do algoritmo, como a ordenação dos dados não se aplica a este contexto, observamos que o gargalo do programa se encontra na leitura das expressões de entrada do programa, visto que as expressões contêm um número n de elementos, e que teremos que verificar todos elementos contidos nas expressões.

CONCLUSÃO

Após o término do programa, conclui-se que as pilhas são estruturas de fácil manipulação de expressões, tornando o programa de fácil implementação. A estrutura de árvore também foi de fácil implementação, visto que, com o auxílio da pilha do próprio compilador e com o uso de recursões, tornou-se portátil a implementação de funções relacionadas à árvore.

Referências

Cruz, A. Árvores `http://equipe.nce.ufrj.br/adriano/c/apostila/arvore.htm`. Acesso em: 14-11-2016.

Preiss, Bruno R. Estruturas de Dados e Algoritmos - Padrões de projetos orientados a objetos com Java. Acesso em: 14-11-2016.

Song, Siang W. Árvore Binária de Busca https://www.ime.usp.br/~song/mac5710/slides/06bst.pdf. Acesso em: 14-11-2016.

Feofiloff, P. Pilhas https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html. Acesso em: 14-11-2016.