Trabalho Prático 0: Notação Polonesa Reversa

Algoritmos e Estruturas de Dados III – 2017/1

Jéssica Taís Carvalho Rodrigues

24 de Abril de 2017

1 Introdução

Existem diversos modos de se calcular operações/expressões , uma delas é pela notação polonesa reversa (RNP - Reverse Polish Notation). Nesta notação, os cálculos são feitos de modo contrário do habitual, como o próprio nome diz, a tabela 1 mostra a diferença em relação notação convencional.
O problema se trata, de receber uma expressão em RNP, mas sem seus operadores, no lugar deles possui um sinal de '?' e também recebe o resultado do cálculo a ser realizado. A partir desses dados, obter todas as possibilidades de operadores, impressos em ordem lexicográfica.
Para que fosse solucionado, ocorre uma simulação do cálculo da expressão dada e onde se encontra as '?'s substitui por um operador (seguindo um ordem lógica) e então no fim o resultado obtido é comparado com o resultado esperado. Caso sejam iguais, os operadores usados formam uma possibilidade de expressão válida.

Notação convencional	Notação Polonesa Reversa
a+b	a b +
(a+b)/c	a b + c/

Tabela 1: Comparação de notações.

2 Solução do problema

O programa conta com três módulos mais o programa principal, sendo eles: pilha, polonesa-reversa e resolucao.
Por dedução, a notação polonesa segue uma lógica recursiva e que intuitivamente pode ser representada por uma pilha.

No módulo da pilha, tem a estrutura de uma pilha comum (pilha de "qualquer coisa", sem tipo definido) e suas respectivas operações para fazê-la funcionar.
No polonesa-reversa, realiza utilizando a pilha, uma calculadora para fazer as operações. É feita de modo recursivo, para que pegue ou o operador de + ou o de *.
No resolucao, se verifica as combinações possíveis e as compara com o resultado dado pela entrada.

3 Análise de complexidade

main: $O(n^2)$, pois é a complexidade máxima dentre todas funções realizadas).

3.1 Tempo

3.1.1 Pilha

Pilha *inicializaPilha(void): O(1), pois possui só declarações e uma comparação, resultando em custo constante;
int pilhaVazia(Pilha *p): O(1), pois só há um retorno para dizer se tem algo na pilha ou não;
void empilha(Pilha *p, void *item): O(1), pois possui só declarações e uma comparação, resultando em custo constante;
void *desempilha(Pilha *p): O(1), pois possui só declarações e uma comparação, resultando em custo constante;
void liberaPilha(Pilha *p): O(1), pois possui só declarações e uma comparação, resultando em custo constante.

3.1.2 Polonesa Reversa

int calculaExp(char *expressao, int resultado): $O(n^2)$, pois no primeiro loop percorre toda a expressão (no pior caso) e depois desempilha que também percorre, sendo n o tamanho da expressão;
int compResultado(char *expressao, int resultado): $O(n^2)$, pois possui retorna calculaExp.

3.1.3 Resolucao

void resolve(char *expressao, int resultado): $O(n^2)$, pois o retorno dela é exp-Possivel que realiza a função de compResultado que também é $O(n^2)$, as demais partes da função possui complexidade menor, logo a que se sobressai é n^2

3.2 Espaço

A complexidade de espaço em todo programa é o tamanho da célula da pilha, sendo que nas funções é igual ao (tamanho da célula) * (tamanho da pilha).

4 Avaliação experimental

Como dito anteriormente a implementação foi realizada de uma maneira e depois alterada para que o tempo fosse otimizado e os resultados do problemas tivesse uma solução melhor.
Para a avaliação experimental, foram realizados testes simples, chamados de testes toys e alguns outros mais complexos, chamados de testes gerados.

4.1 Testes toys

Nestes testes, foram 10 entradas diferentes e a quantidade de operadores estava entre 3 a 7 e os resultados entre 3 e 208. Todos estes executaram em menos de 1 segundo.

4.2 Testes gerados

Nos testes gerados, tinham até 26 operadores e o resultado máximo foi 2090. Até 20 operadores não teve muita diferença entre os tempos, nem entre os algoritmos abordados. Acima de 20 o crescimento foi exponencial, sendo que o normal cresce mais rápido que o otimizado.

4.3 Análise

E importante, observar que o número de operadores interferem diretamente no tempo de execução.
O gráfico 1 ilustra resultados destes testes, tanto para o algoritmo inicial, quanto para o otimizado.

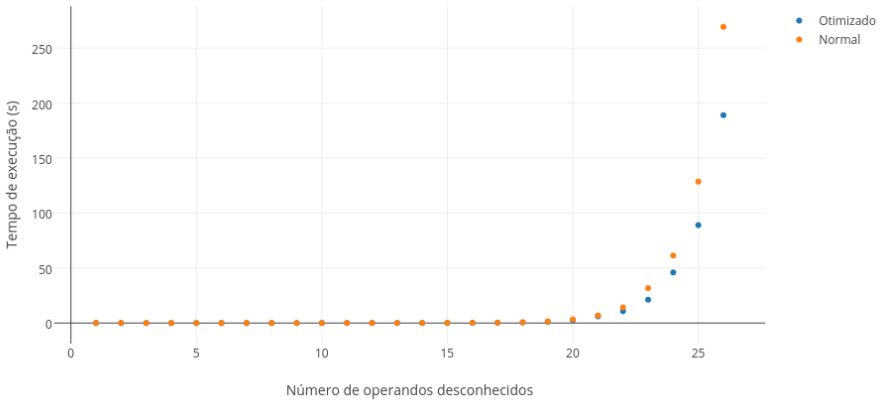

Figura 1: Número de operandos desconhecidos('?') x Tempo de execução

5 Conclusão

Foi um trabalho interessante de realizar, revendo conceitos e aprendendo novos. Tive dificuldade, inicialmente na lógica da notação polonesa reversa pra expressões maiores, mas após pesquisa e prática "na mão", foi esclarecido. E isto foi proveitoso já que facilitou também para assimilar a estrutura da pilha aplicada a ela posteriormente.

6 Referências

- Reverse Polish Notation - Wikipedia. Disponível em `https://en.wikipedia.org/wiki/Reverse_Polish_notation`