# Trabalho Prático 0: Notação Polonesa Reversa

Marcos Paulo Quintao Fernandes

April 2017

## 1 Introdução

Uma Notação Polonesa Reversa, é uma notaçao que pode ser expressa de acordo com a figura 1. Dada uma Notação Polonesa Reversa (de operadores + e * corrompida e o resultado esperado, devemos recuperar todas as sequências de comandos possíveis tal que , após executarmos os comandos, a resposta da notação é igual ao resultado esperado. Basicamente , desenvolveremos um algoritmo de força bruta tal que todas as possibilidades possíveis serão investigadas afim de verificar se alcançamos a solução.

| Notação Convencional | Notação Polonesa Reversa |
|---|---|
| $a + b$ | $ab+$ |
| $(a + b)/c$ | $ab + c/$ |
| $((a + b) * c)/(d - e)$ | $ab + c * de - /$ |

1 - Esquema da notaçao

## 2 Solução do Problema

Dada uma entrada válida, ou seja, para todo prefixo existem mais números inteiros que operadores, iremos avaliar as soluções da seguinte forma:

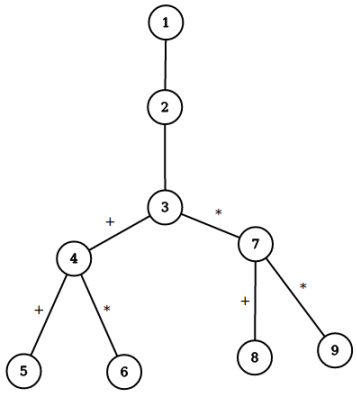
$f(i)$ = pilha que corresponde à Notação Polonesa Reversa lida até o momento da leitura do i-ésimo elemento da entrada. $op(i)$ = pilha que corresponde às operações utilizadas para chegarmos à pilha f(i).

1. $f(0)$ = vazio, $op(0)$ = vazio.
2. Se o i-ésimo elemento for um número, temos que $f(i+1) = f(i)+$ último número ; $op(i) = op(i-1)$.
3. Se o i-ésimo elemento for um operador, temos que ou $f(i+1) = f(i-3)+$ soma dos 2 números do topo da pilha de $f(i)$ e $op(i) = op(i-1)+$ '+' ou então $f(i+1) = f(i-3)+$ multiplicação dos 2 números do topo da pilha de $f(i)$ e $op(i) = op(i-1)+$ '*'.

Note que se o i-ésimo elemento da entrada for um operador, temos que o $|f(i)| = |f(i-1)| - 1$.

Como só utilizaremos os operadores de + e de *, temos que se o valor do topo da fila for maior que o valor esperado, temos que a solução a ser montada já está errada. Desta forma conseguimos visitar menos estados inválidos e como veremos a seguir, isso otimizará muito a solução.

Para o input igual a:
2 2 2 ? ?
8
temos:




2 - Esquema do algoritmo

Para o esquema acima temos:

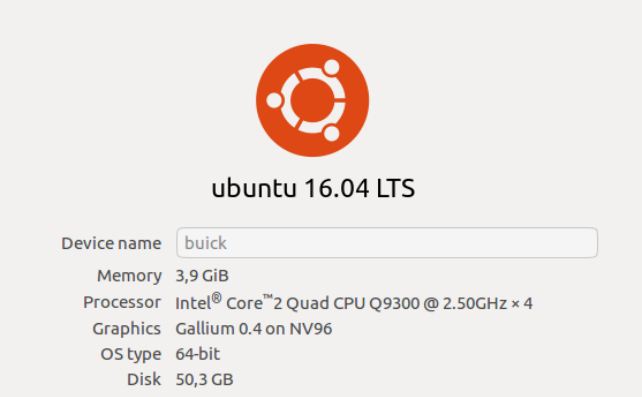
| | |
|---|---|
| f(1) = 2 | op(1) = vazio; |
| f(2) = 2 2 | op(2) = vazio |
| f(3) = 2 2 2 | op(3) = vazio |
| f(4) = 2 4 | op(4) = + |
| f(5) = 6 | op(5) = + + |
| f(6) = 8 | op(6) = + '*' |
| f(7) = 2 4 | op(7) = * |
| f(8) = 6 | op(8) = * + |
| f(9) = 8 | op(9) = * * |

# 3 Análise de Complexidade

A princípio, a complexidade da solução é $O(2^N)$, pois a cada iteração do tipo ? são chamadas 2 funções e existem $\lfloor \frac{N}{2} \rfloor$ operadores do tipo ?. No entanto, conseguimos uma solução um pouco melhor do que isso uma vez que não serão visitados todos os estados da recursão, já que muitos deles serão inválidos. Por exemplo, para os casos em que todos os elementos da entrada são maiores que 1 e que a resposta esperada é um inteiro de 32 bits, temos que é impossivel multiplicarmos mais que 32 vezes. Em outras palavras, todos os estados que utilizam mais que 32 multiplicações, não serão visitados.

# 4 Avaliação Experimental

Para avaliarmos o desempenho desse programa utilizaremos a uma máquina com as seguintes configurações:




3 - Configurações da Máquina

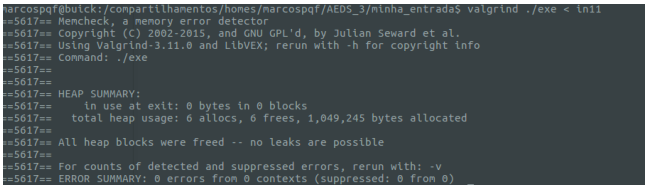
Foi testado o programa para diversas instâncias. Não foi possível testar par instâncias maiores porque me faltou tempo.

Cada teste, foi analizado para uma resposta $2^{n-2}$ sendo n o numero de interrogações da instância. Segue abaixo uma tabela de desempenho dos testes:

Por fim , para analisar se houve algum leaking de memória, rodamos o valgrind em um subconjunto dos testes, obtivemos em todos eles. Segue abaixo um dos resultados do valgrind:

Table 1: tamanho da entrada x desempenho

| TAMANHO DA INSTANCIA(N) | TEMPO DECORRIDO(s) |
|---|---|
| 5 | 0.0057 |
| 7 | 0.0087 |
| 9 | 0.0039 |
| 11 | 0.0034 |
| 13 | 0.0042 |
| 15 | 0.0052 |
| 17 | 0.0045 |
| 19 | 0.0046 |
| 29 | 0.1076 |
| 39 | 0.2087 |
| 49 | 5.2315 |
| 59 | 165.86 |



4 - Resultado do Valgrind