Trabalho Prático 1

Expressões lógicas e satisfabilidade

Raquel Gonçalves Rosa - 2020105815

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

raquelgr@ufmg.br

1. Introdução

Neste trabalho, foram propostas a implementação de duas formas de lidar com problemas lógicos: avaliação de uma expressão e o clássico problema de satisfabilidade. Para a implementação foram utilizados como base os códigos de Pilhas Encadeadas, para a avaliação, e os códigos de Árvores, para a satisfabilidade, ambos disponibilizados nos slides da disciplina. Tendo em vista isso, este documento possui o objetivo de explicitar a implementação de soluções para os problemas propostos.

2. Método

2.1 Implementação

O programa foi desenvolvido em C++ com compilador G++, em ambiente Linux.

2.2 Estrutura de dados

2.2.1 Avaliação de uma expressão lógica

Para implementação do programa foi utilizada a estrutura de dados Pilha Encadeada juntamente com os conhecimentos adquiridos sobre a notação pós-fixa, também conhecida como notação polonesa. Essa estrutura foi implementada na classe **Pilha** (arquivo pilha.cpp) e utilizada como auxiliar na classe **Avaliacao** (arquivo avaliacao.cpp).

A classe **Avaliacao,** responsável por resolver todo o problema da avaliação de expressões, está estruturada nos seguintes métodos:

- **Métodos privados**
    - Avaliacao::RemoverEspacos
    - Avaliacao::ParentesesBalanceados
    - Avaliacao::UltimoCaractereValido
    - Avaliacao::CaracteresValidos
    - Avaliacao::ConverterInfixaParaPosfixa
- **Método público**
    - Avaliacao::Avaliar

Nos estudos para a realização da resolução, foi identificada uma forma bem ágil e simples de se resolver: a notação polonesa. Essa notação foi proposta pelo matemático polonês Jan Lukasiewicz, como uma forma de escrever expressões aritméticas e lógicas sem a necessidade de utilizar parênteses, ela também organiza os operadores por sua ordem de prioridade. A implementação de uma pilha é a mais recomendada e é a que foi utilizada para realizar a conversão da fórmula na notação infixa para a notação polonesa. Sua implementação está contida no método **ConverterInfixaParaPosfixa,** além do mais, a busca e atribuição da valoração da posição também é feita nesse método.

Após a conversão, o método **Avaliar**, que também utiliza uma pilha, percorre a fórmula pós-fixa da esquerda para a direita com o seguinte comportamento:

→ Se é dígito, empilha.
→ Se é operador de negação (~), desempilha um número, faz sua conversão e o empilha novamente.
→ Se é operador de conjunção (&) ou disjunção (|), desempilha dois números e realiza a operação entre eles e empilha o resultado.

Ao final, sobra apenas um número empilhado, sendo este o resultado da avaliação.

Importante ressaltar, que para ambas as classes foi utilizada uma biblioteca, implementada na classe auxiliar **Funcoes** (arquivo funcoes.cpp), para verificar se era dígito, negação, conjunção ou disjunção e verificar a prioridade entre os operadores.

2.2.2 Satisfabilidade

Para implementação do programa foi utilizada a estrutura de dados Árvore, implementada na classe **ArvoreDeSatisfabilidade** (arquivo arvoreDeSatisfabilidade.cpp).

A classe **ArvoreDeSatisfabilidade,** responsável por resolver o problema de satisfabilidade, está estruturada nos seguintes métodos:

- **Métodos privados**
    - ArvoreDeSatisfabilidade::Insere
    - ArvoreDeSatisfabilidade::InsereRecursivo
    - ArvoreDeSatisfabilidade::CaminhaEResolve
    - ArvoreDeSatisfabilidade::Limpa
    - ArvoreDeSatisfabilidade::ApagaRecursivo
- **Método publico**
    - ArvoreDeSatisfabilidade::VerificarSatisfablidade

O método **InsereRecursivo** recebe a valoração, que passa a ser a raiz da árvore, para cada variável presente, podendo ser 'e' (∃) ou 'a' (∀), é criado dois filhos, um com atribuição do valor 0 para essa variável e outro com a atribuição do valor 1. Após criar os dois primeiros filhos para a primeira variável, é verificado se em cada um deles ainda tem outras variáveis, a inserção se dá de forma recursiva até não restarem mais variáveis nos nós folha.

O método **CaminhaEResolve** recebe a fórmula e a raiz da árvore criada no método explicitado acima, nela é feito um caminhamento Pós-Ordem de forma recursiva, caso seja um nó folha sabemos que não há variáveis presente na sua valoração, dessa forma basta chamar a classe **Avaliacao** para realizar a avaliação de uma expressão simples. Caso contrário, o método faz uma extensa avaliação, que não cabe ser totalmente explicitada neste documento, mas que, em linhas gerias, se dá por meio da verificação do tamanho do item presente nos filhos e a variável do nó analisado no momento, esses dados passam por várias estruturas condicionais para verificar a substituição necessária para o nó pai. Essa análise é feita até chegar no nó pai, seguindo a forma de caminhamento pós-ordem. Ao final do caminhamento, sabemos que o resultado está na raiz da árvore.

A árvore criada tem altura n e 2^n nós folha, sendo n a quantidade de variáveis presentes na fórmula.

2.3 Organização

O programa está dividido em:

- 4 cabeçalhos, contidos na pasta *include*, sendo **arvoreDeSatisfabilidade.hpp**, **avaliacao.hpp**, **funcoes.hpp** e **pilha.hpp**.
- 5 arquivos de código, contidos na pasta *src*, sendo **arvoreDeSatisfabilidade.cpp**, **avaliacao.cpp**, **funcoes.cpp**, **pilha.cpp** e o **main.cpp**, sendo o último responsável pela leitura do comando passado via terminal e chamada das funções necessárias.

3. Análise de Complexidade

A seguir será realizada a análise de complexidade de tempo de todas os métodos implementados no projeto, não será analisada a complexidade dos métodos disponibilizados nos slides da disciplina e que não foram modificados.

Classe **Funcoes:**

- Todas os métodos dessa classe são $O(1)$ pois são métodos de verificação simples, utilizando apenas estruturas condicionais.

Classe **Avaliacao:**

- Avaliacao::RemoverEspacos: $O(n)$, sendo n o tamanho da string, o método sempre percorre toda ela.
- Avaliacao::ParentesesBalanceados: $O(n)$, sendo n o tamanho da string, o método sempre percorre toda ela.
- Avaliacao::UltimoCaractereValido: $O(1)$, avalia apenas o último caractere.
- Avaliacao::CaracteresValidos: $O(n)$, sendo n o tamanho da string, o método sempre percorre toda ela.
- Avaliacao::ConverterInfixaParaPosfixa: $O(n)$, sendo n o tamanho da string, o método sempre percorre toda ela.
- Avaliacao::Avaliar: $O(n)$, sendo n o tamanho da string, o método sempre percorre toda ela.

Classe **ArvoreDeSatisfabilidade**

- ArvoreDeSatisfabilidade::InsereRecursivo: a árvore segue o padrão de uma árvore binária balanceada, portanto, temos o melhor caso de inserção em uma árvore, $O(\log\ n)$, sendo n a quantidade de nós.
- ArvoreDeSatisfabilidade::CaminhaEResolve: nesse caso, além de caminhar pela árvore são feitas várias operações condicionais todas elas limitadas por um for que percorre de 0 até n , sendo n o tamanho da valoração, logo em cada nó temos uma complexidade de $O(n)$. Sabemos que pra percorrer a complexidade é $O(\log\ n)$ por ser uma árvore binária equilibrada. Portanto, a complexidade total é $O(n \cdot \log\ n)$.

4. Estratégias de Robustez

Algumas verificações foram implementadas para atender as especificações, todas lançam exceções específicas. São elas:

- Opção escolhida inválida;
- Tamanho máximo da fórmula (10^6) e da valoração (100)
- Balanceamento de parênteses;
- Último caractere valido;
- Caracteres permitidos;
- Existência da posição do digito na valoração;
- Valores inválidos para as operados de negação, conjunção e disjunção;
- Variáveis permitidas na valoração;
- Pilha vazia;

Existem duas funções especiais que foram criadas visando o encapsulamento, sendo elas: **Avaliar** e **VerificarSatisfablidade,** a estratégia foi pensada dessa forma para não complicar o código do **main**.

5. Análise Experimental

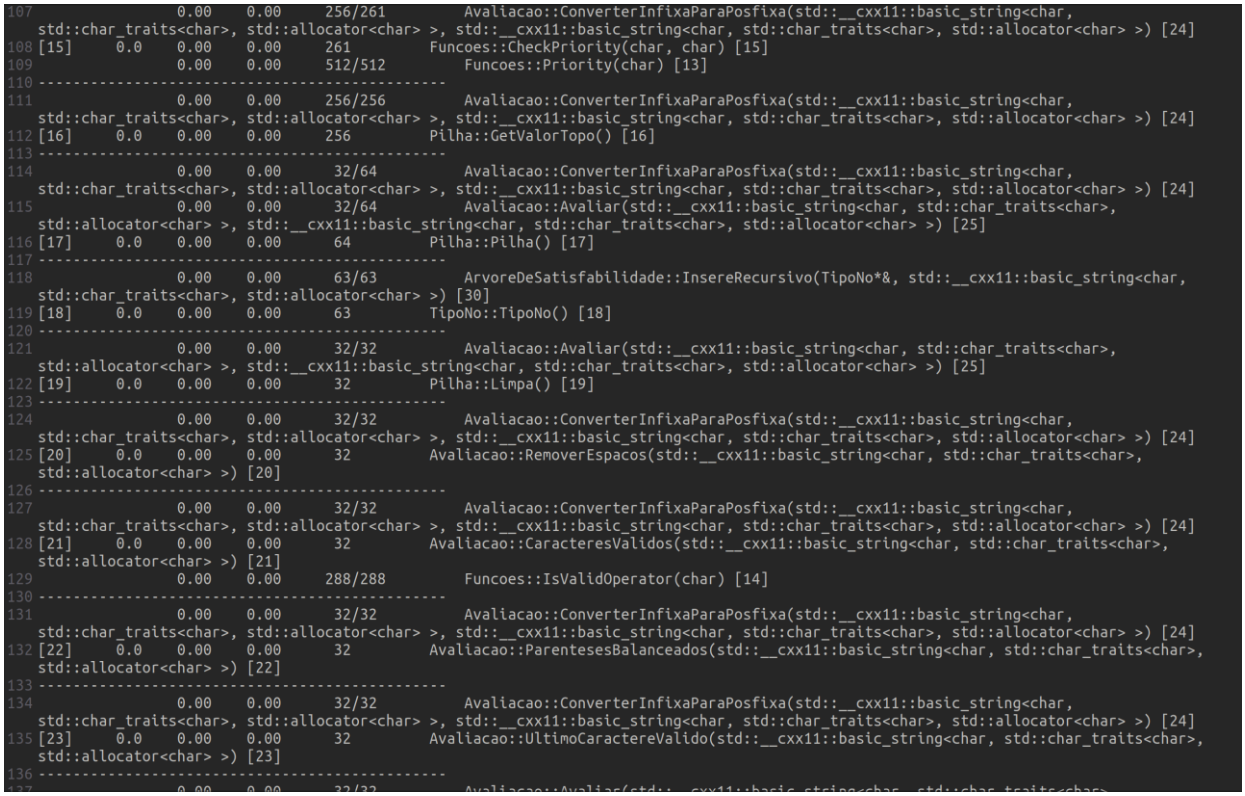
Utilizando a ferramenta *gprof* com a entrada:

-s "0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9" e1e1e1e1e1

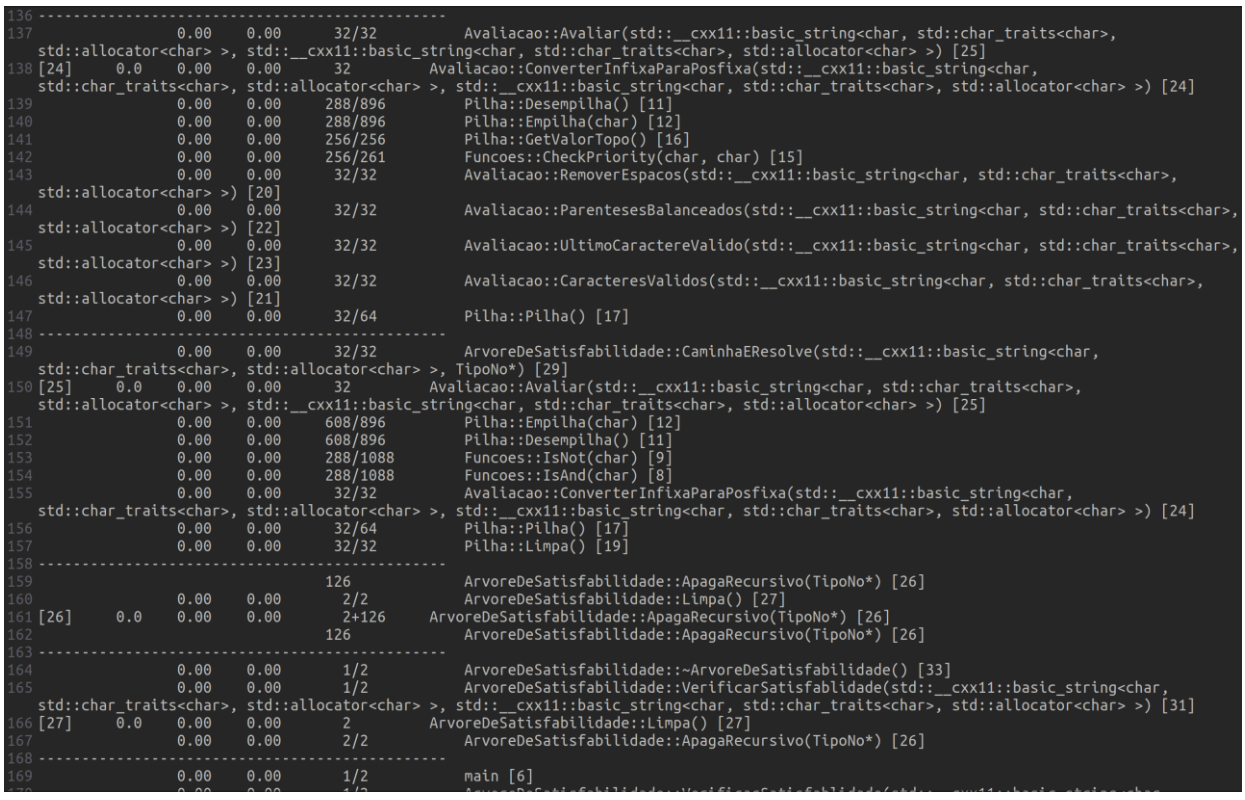
Foram obtidos os seguintes resultados:

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00     1088     0.00     0.00  Funcoes::IsAnd(char)
  0.00      0.00     0.00     1088     0.00     0.00  Funcoes::IsNot(char)
  0.00      0.00     0.00      896     0.00     0.00  TipoCelula::TipoCelula()
  0.00      0.00     0.00      896     0.00     0.00  Pilha::Desempilha()
  0.00      0.00     0.00      896     0.00     0.00  Pilha::Empilha(char)
  0.00      0.00     0.00      512     0.00     0.00  Funcoes::Priority(char)
  0.00      0.00     0.00      288     0.00     0.00  Funcoes::IsValidOperator(char)
  0.00      0.00     0.00      261     0.00     0.00  Funcoes::CheckPriority(char, char)
  0.00      0.00     0.00      256     0.00     0.00  Pilha::GetValorTopo()
  0.00      0.00     0.00       64     0.00     0.00  Pilha::Pilha()
  0.00      0.00     0.00       63     0.00     0.00  TipoNo::TipoNo()
  0.00      0.00     0.00       32     0.00     0.00  Pilha::Limpa()
  0.00      0.00     0.00       32     0.00     0.00  Avaliacao::RemoverEspacos(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >)
  0.00      0.00     0.00       32     0.00     0.00  Avaliacao::CaracteresValidos(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00       32     0.00     0.00  Avaliacao::ParentesesBalanceados(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00       32     0.00     0.00  Avaliacao::UltimoCaractereValido(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00       32     0.00     0.00  Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00       32     0.00     0.00  Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00        2     0.00     0.00  ArvoreDeSatisfabilidade::ApagaRecursivo(TipoNo*)
  0.00      0.00     0.00        2     0.00     0.00  ArvoreDeSatisfabilidade::Limpa()
  0.00      0.00     0.00        2     0.00     0.00  ArvoreDeSatisfabilidade::ArvoreDeSatisfabilidade()
  0.00      0.00     0.00        1     0.00     0.00  ArvoreDeSatisfabilidade::CaminhaEResolve(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >, TipoNo*)
  0.00      0.00     0.00        1     0.00     0.00  ArvoreDeSatisfabilidade::InsereRecursivo(TipoNo*&, std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00        1     0.00     0.00  ArvoreDeSatisfabilidade::VerificarSatisfablidade(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00        1     0.00     0.00  ArvoreDeSatisfabilidade::Insere(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >)
  0.00      0.00     0.00        1     0.00     0.00  ArvoreDeSatisfabilidade::~ArvoreDeSatisfabilidade()
```

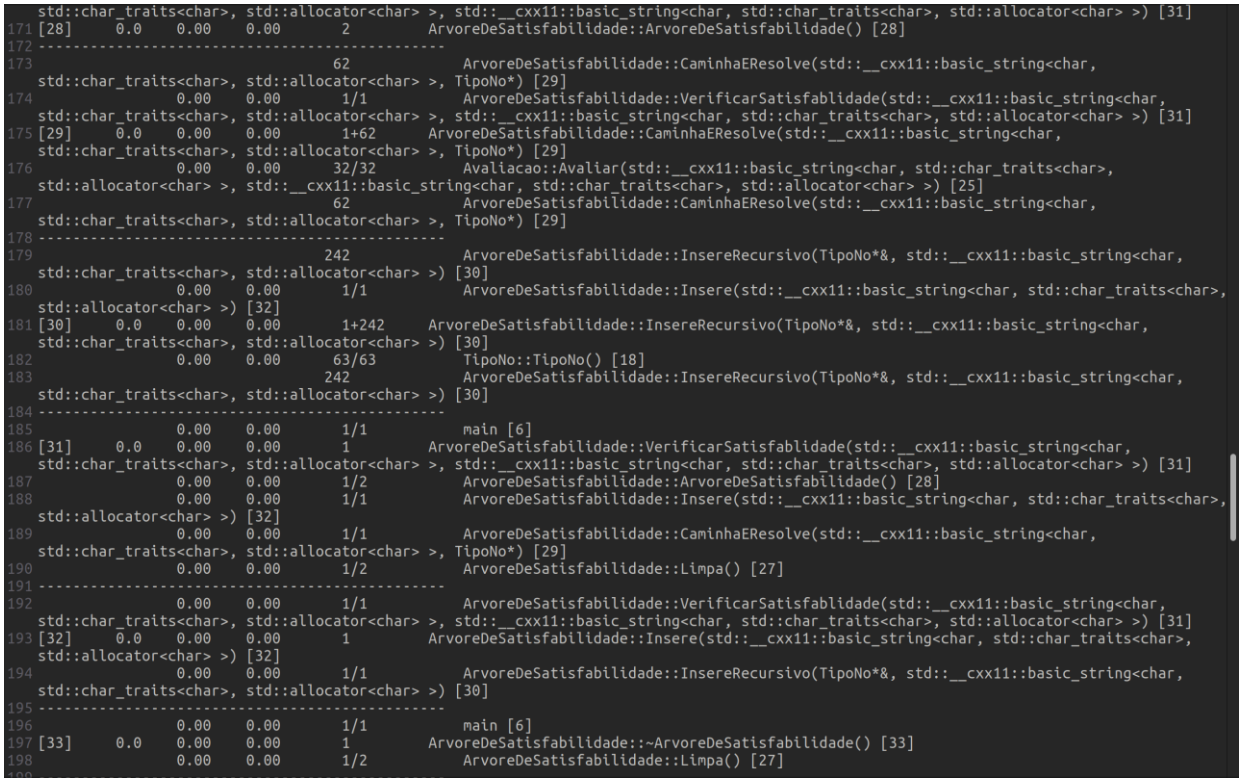
```
granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00     288/1088        Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
                0.00    0.00     288/1088        Funcoes::IsValidOperator(char) [14]
                0.00    0.00     512/1088        Funcoes::Priority(char) [13]
[8]      0.0    0.00    0.00    1088         Funcoes::IsAnd(char) [8]
-----------------------------------------------
                0.00    0.00     288/1088        Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
                0.00    0.00     288/1088        Funcoes::IsValidOperator(char) [14]
                0.00    0.00     512/1088        Funcoes::Priority(char) [13]
[9]      0.0    0.00    0.00    1088         Funcoes::IsNot(char) [9]
-----------------------------------------------
                0.00    0.00     896/896         Pilha::Empilha(char) [12]
[10]     0.0    0.00    0.00     896         TipoCelula::TipoCelula() [10]
-----------------------------------------------
                0.00    0.00     288/896         Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
                0.00    0.00     608/896         Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
[11]     0.0    0.00    0.00     896         Pilha::Desempilha() [11]
-----------------------------------------------
                0.00    0.00     288/896         Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
                0.00    0.00     608/896         Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
[12]     0.0    0.00    0.00     896         Pilha::Empilha(char) [12]
                0.00    0.00     896/896         TipoCelula::TipoCelula() [10]
-----------------------------------------------
                0.00    0.00     512/512         Funcoes::CheckPriority(char, char) [15]
[13]     0.0    0.00    0.00     512         Funcoes::Priority(char) [13]
                0.00    0.00     512/1088        Funcoes::IsNot(char) [9]
                0.00    0.00     512/1088        Funcoes::IsAnd(char) [8]
-----------------------------------------------
                0.00    0.00     288/288         Avaliacao::CaracteresValidos(std::__cxx11::basic_string<char, std::char_traits<char>,
 std::allocator<char> >) [21]
[14]     0.0    0.00    0.00     288         Funcoes::IsValidOperator(char) [14]
                0.00    0.00     288/1088        Funcoes::IsNot(char) [9]
                0.00    0.00     288/1088        Funcoes::IsAnd(char) [8]
-----------------------------------------------
                0.00    0.00       5/261         main [6]
                0.00    0.00     256/261         Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
```



```
                0.00    0.00     256/261         Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
[15]     0.0    0.00    0.00     261         Funcoes::CheckPriority(char, char) [15]
                0.00    0.00     512/512         Funcoes::Priority(char) [13]
-----------------------------------------------
                0.00    0.00     256/256         Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
[16]     0.0    0.00    0.00     256         Pilha::GetValorTopo() [16]
-----------------------------------------------
                0.00    0.00      32/64          Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
                0.00    0.00      32/64          Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
[17]     0.0    0.00    0.00      64         Pilha::Pilha() [17]
-----------------------------------------------
                0.00    0.00      63/63          ArvoreDeSatisfabilidade::InsereRecursivo(TipoNo*&, std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >) [30]
[18]     0.0    0.00    0.00      63         TipoNo::TipoNo() [18]
-----------------------------------------------
                0.00    0.00      32/32          Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
[19]     0.0    0.00    0.00      32         Pilha::Limpa() [19]
-----------------------------------------------
                0.00    0.00      32/32          Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
[20]     0.0    0.00    0.00      32         Avaliacao::RemoverEspacos(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [20]
-----------------------------------------------
                0.00    0.00      32/32          Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
[21]     0.0    0.00    0.00      32         Avaliacao::CaracteresValidos(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [21]
                0.00    0.00     288/288         Funcoes::IsValidOperator(char) [14]
-----------------------------------------------
                0.00    0.00      32/32          Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
[22]     0.0    0.00    0.00      32         Avaliacao::ParentesesBalanceados(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [22]
-----------------------------------------------
                0.00    0.00      32/32          Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
[23]     0.0    0.00    0.00      32         Avaliacao::UltimoCaractereValido(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [23]
-----------------------------------------------
```



```
-----------------------------------------------
                0.00    0.00      32/32          Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
[24]     0.0    0.00    0.00      32         Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
                0.00    0.00     288/896         Pilha::Desempilha() [11]
                0.00    0.00     288/896         Pilha::Empilha(char) [12]
                0.00    0.00     256/256         Pilha::GetValorTopo() [16]
                0.00    0.00     256/261         Funcoes::CheckPriority(char, char) [15]
                0.00    0.00      32/32          Avaliacao::RemoverEspacos(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [20]
                0.00    0.00      32/32          Avaliacao::ParentesesBalanceados(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [22]
                0.00    0.00      32/32          Avaliacao::UltimoCaractereValido(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [23]
                0.00    0.00      32/32          Avaliacao::CaracteresValidos(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >) [21]
                0.00    0.00      32/64          Pilha::Pilha() [17]
-----------------------------------------------
                0.00    0.00      32/32          ArvoreDeSatisfabilidade::CaminhaEResolve(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, TipoNo*) [29]
[25]     0.0    0.00    0.00      32         Avaliacao::Avaliar(std::__cxx11::basic_string<char, std::char_traits<char>,
  std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [25]
                0.00    0.00     608/896         Pilha::Empilha(char) [12]
                0.00    0.00     608/896         Pilha::Desempilha() [11]
                0.00    0.00     288/1088        Funcoes::IsNot(char) [9]
                0.00    0.00     288/1088        Funcoes::IsAnd(char) [8]
                0.00    0.00      32/32          Avaliacao::ConverterInfixaParaPosfixa(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [24]
                0.00    0.00      32/64          Pilha::Pilha() [17]
                0.00    0.00      32/32          Pilha::Limpa() [19]
-----------------------------------------------
                              126             ArvoreDeSatisfabilidade::ApagaRecursivo(TipoNo*) [26]
                0.00    0.00       2/2           ArvoreDeSatisfabilidade::Limpa() [27]
[26]     0.0    0.00    0.00       2+126     ArvoreDeSatisfabilidade::ApagaRecursivo(TipoNo*) [26]
                              126             ArvoreDeSatisfabilidade::ApagaRecursivo(TipoNo*) [26]
-----------------------------------------------
                0.00    0.00       1/2           ArvoreDeSatisfabilidade::~ArvoreDeSatisfabilidade() [33]
                0.00    0.00       1/2           ArvoreDeSatisfabilidade::VerificarSatisfablidade(std::__cxx11::basic_string<char,
  std::char_traits<char>, std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) [31]
[27]     0.0    0.00    0.00       2         ArvoreDeSatisfabilidade::Limpa() [27]
                0.00    0.00       2/2           ArvoreDeSatisfabilidade::ApagaRecursivo(TipoNo*) [26]
-----------------------------------------------
                0.00    0.00       1/2           main [6]
```



Como o esperado, o método **Avaliar** em conjunto com a **ConverterInfixaParaPosfixa** foi chamada 32 vezes, como o teste tem 5 variáveis, era esperado que fossem criados 2^5 nós folha. Pode-se notar também que o método **InsereRecursivo** foi chamado 63 vezes, isso se refere a quantidade de nós criados, tal quantidade pode ser explicada pela fórmula $k = 2^{n+1} - 1$, onde k é a quantidade de nós e n e a quantidade de variáveis e por fim método **CaminhaEResolve** foi chamado 62 vezes porque analisa todos os nós e termina no nó raiz.

A diferença crucial, é que principal método da avaliação tem complexidade $O(n)$ e o principal da satisfabilidade tem complexidade $O(n \cdot \log \quad n)$. Tendo que a satisfabilidade sempre chama a avalição, nota-se para o limite estabelecido de 5 variáveis foram feitas 32 chamadas para a avaliação e 63 para a satisfabilidade, ou seja, elas crescem exponencialmente de acordo com a quantidade de variáveis.

6. Conclusões

A implementação dos problemas põe em prática os conteúdos vistos em sala, e é possível notar que apesar da árvore binária ter uma implementação mais complexa, por ter uma ordem de complexidade computacional mais baixa, ela se torna muito eficiente.

7. Bibliografia

CHAIMOWICZ, L; PRATES, R. Pilhas e Filas. 2020. Apresentação do Power Point.

CHAIMOWICZ, L; PRATES, R. Árvores. 2019. Apresentação do Power Point.

Evaluation of Postfix Expression. Disponível em: <https://www.geeksforgeeks.org/evaluation-of-postfix-expression/>. Acesso em: 15 out. 2023.

Convert Infix expression to Postfix expression. Disponível em: <https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression/>. Acesso em: 15 out. 2023.

Notação polonesa — Exercícios-programa de MAC0122. Disponível em: <https://panda.ime.usp.br/algoritmos/static/eps/ep6/parted/parted-polonesa.html#:~:text=A%20nota%C3%A7%C3%A3o%20Polonesa%20(reversa)%20ou>. Acesso em: 15 out. 2023.