INF1413 Teste de Software
Período: 2018-1
Profs. Arndt von Staa

4o. Trabalho

Data de divulgação: 09 de junho (sábado)

Data de entrega: 19 de junho (terça-feira)

1. Descrição do trabalho

O trabalho simula um trabalho de manutenção, utilizando técnicas formais, execução e geração automatizada de massas de teste, e uso de mutantes. O trabalho utiliza um programa incompleto e contendo alguns defeitos. O programa já está instrumentado. O ambiente de execução é o sistema de teste automatizado **Talisman**, voltado para C++. O programa executa em uma janela de linha de comando do **windows (cmd)**. Foi desenvolvido e testado com Windows 10. O programa a ser testado cria árvores **AVL**. O algoritmo de inserção é baseado no encontrado em

//www.geeksforgeeks.org/avl-tree-set-1-insertion/

Foram acrescentados comandos para manter a referência "pParent" de cada elemento inserido na árvore. Foi acrescentado também um verificador estrutural, que, assim espero, está correto e completo. Além disso o código do site, foi transformado em um código seguindo os padrões do projeto Talisman.

Utilizo uma sub-linguagem de C++, que tem um forte sabor C. O log gerado pode ser apresentado na console, ou ser transferido para um arquivo.

2. Descrição do quarto trabalho

2.1. Preparação:

Expanda o arquivo zipado para o diretório que será utilizado pelo trabalho. Entre no subdiretório "**batches**". Execute "**compile tst-avl**". É possível que apareça um erro de biblioteca. São fornecidas três bibliotecas na pasta "**obj**". A biblioteca **TalismanTestLib.lib** é a biblioteca utilizada para compilar o programa. Infelizmente ela depende da versão do compilador. Eu uso compiladores C++ do Visual Studio. No diretório encontram-se também as bibliotecas **TalismanTestLib1800.lib** e **TalismanTestLib1900.lib**. Copie a que o compilador espera para o arquivo **TalismanTestLib.lib**. Se o erro de biblioteca persistir, será necessário recompilar a biblioteca. O programa da ferramenta de teste do **Talisman** e seus **scripts** de teste estão disponíveis na aba **software** do site da disciplina. Vem junto um texto explicando como proceder para compilar.

1. Execute o teste (linha de comando) **test avl 01**. Deveria rodar até o final. Ao executar serão capturadas e ignoradas diversas exceções de sistema, correspondentes a erros do sistema.
2. A documentação do módulo de teste genérico encontra-se no arquivo **GNRCTEST.HPP**. Como operar o sistema encontra-se no arquivo **MAIN.HPP**.

2.2. Trabalho:

1. Teste o programa com o arquivo **tst-avl-01.script**. Desenhe a árvore que não deu erro. Para cada caso de teste que deu erro (em geral uma exceção de sistema) mostre o estado da árvore antes de inserir o elemento que dará erro, bem como o elemento, que ao ser inserido, provoque o erro.

O **display** de uma árvore ocorre em ordem **infix**. Cada nó exibido contém a altura da sub-árvore da qual é raiz, a identificação do nó pai e a identificação do nó.

2. Desenhe as árvores geradas até o momento do erro pelos scripts **tst-avl-02.script** e **tst-avl-03.scipt**. Indique no desenho qual o elemento que seria adicionado caso não tivesse ocorrido erro.

3. Corrija o programa. Desenhe o grafo de chamada a partir do método **InsertAVLtreeSymbol(char * pSymbol)** restrito ao módulo **AVLTREE**. Os erros do programa fornecido se concentram nos métodos desse grafo de chamada. Ao fazer a correção, anote em um arquivo, as ações que você realizou, bem como a razão de realiza-las e os respectivos resultados.Caso não consiga corrigir mostre, de forma convincente, o porquê não ter conseguido. O texto deve estar em ordem temporal de execução das ações. **OBS. não ter conseguido corrigir, de qualquer forma perde 1,5 na nota.**

4. Insira dois mutantes no método **DoInsertAVLelement**. Retire do script os casos que geram erros de execução. Verifique se o teste do caso "**// Árvore que não dá erro**" é capaz de matar os mutantes. Não sendo, explique o porquê, e, a seguir, mostre o **script** de teste ampliado que é capaz de mata-los. Caso tenha corrigido o programa, item 3, execute o teste utilizando a geração de casos de teste aleatórios (ver item 5).

5. Complete o interpretador de teste usando o gerador de casos de teste. Veja o comando **GenerateTest** que se encontra no executor de testes **TST_AVL.CPP**. Complete o interpretador substituindo os comandos **dummy**. Considerando os 30 primeiros elementos do vetor **vtAVLsymbols** (está no módulo **TST-AVL**), gere 35 execuções de **DoInsertSymbol** usando seleções aleatórias dentro do conjunto de 30 símbolos. Caso a inserção ocorra verifique se, era **false** o valor do respectivo **isInTree**, troque-o para **true**. Não sendo **false** reporte um erro. Caso a inserção não ocorra, verifique se, era **false** o valor do respectivo **isInTree**. Reporte um erro caso seja **false**. Capture as exceções. Reporte todos os erros por ventura encontrados. Caso não tenha conseguido corrigir o programa (item 3) use o programa em erro.

3. Entrega do Trabalho

O trabalho deve ser realizado em grupos de 2 ou 3 alunos.

Devem ser entregues **os códigos fonte e os executáveis dos programas.** Deve ser entregue também um arquivo **.DOC** (ou **.DOCX**) contendo as observações e comentários para cada um dos itens acima, em especial o relato do trabalho realizado para (tentar) corrigir. Após gerar o **.zip** (ou **.rar**) a ser entregue, rebatize-o para **.zipx** (ou **.rarx**) para que o sistema de correio eletrônico não bloqueie a transmissão.

Além dos arquivos acima, deve ser entregue um relatório de tempo despendido no trabalho. O relatório consta de uma tabela de registro de trabalho do grupo organizada como a seguir:

Data Horas Trabalhadas Quem trabalhou Tipo Tarefa Descrição da Tarefa Realizada

A coluna *Quem trabalhou* identifica 1 ou mais membros do grupo que realizaram a tarefa. Na descrição da tarefa redija uma explicação breve sobre o que foi feito. Esta descrição deve estar de acordo com o Tipo Tarefa. Cada Tipo Tarefa identifica uma natureza de atividade que deverá ser discriminada explicitamente, mesmo que, durante uma mesma sessão de trabalho tenham sido realizadas diversas tarefas. **Crie e justifique a lista de naturezas de tarefas**.

4. Critérios de correção básicos

As avaliações dos itens resultam em { OK, +/- -0,5, fraco -1, ruim -2, não fez -3 }. A soma dos descontos por item pode, obviamente, resultar em um número negativo.

A nota da parte "trabalho" leva em conta a frequência dos conceitos.

A nota final leva em conta a nota do trabalho descontando as penalidades por não obedecer aos padrões publicados.

Será considerada a clareza, a ortografia e a sintaxe dos textos.