Universidade de São Paulo

ACH2023 ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Ivandré Paraboni ivandre@usp.br

Semestre 01/2009

Trabalho Prático 2 – Árvore de Busca Binária

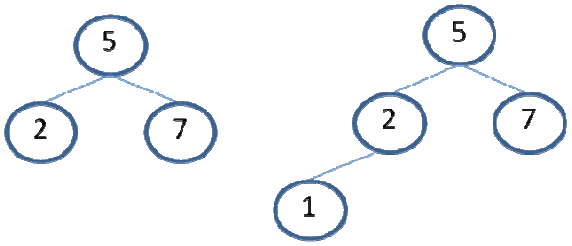
Descrição: A partir do projeto exemplo disponibilizado no COL (*epArvore.dev*), editar o módulo *trabalho.cpp* implementando as seguintes operações sobre uma estrutura de dados do tipo árvore de busca binária:

1. A estrutura a ser manipulada é uma árvore de busca binária de implementação dinâmica contendo inteiros.
2. O projeto exemplo fornecido contém uma função *main.cpp* que pode ser usada para execução e teste do seu programa, um header *aed1b.h* contendo os tipos de dados manipulados e outras especificações globais, e o módulo *trabalho.cpp* onde deve ser realizado o trabalho propriamente dito.
3. O objetivo do trabalho é implementar de forma correta e completa as 3 funções a seguir, que devem estar inteiramente contidas no módulo *trabalho.cpp*:

   - **`void manutencao(NO* *raiz, int ch)`**
     Se a chave *ch* não existir na árvore apontada por **raiz*, *ch* é inserida na posição correta. Se *ch* já existir, o nó correspondente deve ser removido da estrutura.

   - **`NO *sucessor(NO *raiz, int n)`**
     Retorna um ponteiro para o nó que contem a primeira chave após o valor *n* na árvore apontada por **raiz* na ordem crescente de valores de chave. Por exemplo, dada a estrutura {1,3,5,6,8,10} e *n*=3, a função retorna um ponteiro para o nó 5. Já para *n*=6 é retornado um ponteiro para 8, e para *n*=10 (ou qualquer *n* inexistente) retorna *NULL*.
     Importante: não use vetores ou listas auxiliares nesta implementação.

   - **`int folhaMaisProxima(NO *raiz)`**
     Retorna o número da chave contida na folha mais próxima da raiz, ou seja, no nível mais alto possível. Havendo duas folhas no mesmo nível deve ser considerada prioritariamente a mais a esquerda. Por exemplo, na árvore à esquerda abaixo, a folha mais próxima da raiz é 2, e na árvore à direita é 7.



4. Note que para testar a sua implementação e garantir sua correção você provavelmente terá de criar várias outras funções auxiliares (e.g., entrada e exibição de dados, percursos, busca etc.) que não fazem parte do trabalho propriamente dito e que não serão avaliadas.
5. Tenha em mente que tudo que for necessário para executar as funções solicitadas deve obrigatoriamente estar em *trabalho.cpp*, ou seu EP estará incompleto.
6. Além das 3 funções acima, solicita-se que o aluno complete as funções *aluno, turma* e *nrousp* também presentes no módulo *trabalho.cpp* para fins de identificação do autor.

7. IMPORTANTE: o seu programa será corrigido de forma *automática*, e por isso você não pode alterar as assinaturas das funções solicitadas, nem os tipos de dados ou especificações no *header aed1b.h* que acompanha o projeto exemplo. Pelo mesmo motivo, caso você não implemente alguma das funções solicitadas deve mantê-la com o código vazio em *trabalho.cpp* para que o programa possa ser compilado.
8. Restrições de implementação: seu programa deve ser *muito eficiente*. Em especial, não execute repetidamente funções que percorrem a estrutura inteira, ou que buscam informação que o algoritmo teria meios de acessar sem nova busca.

Modalidade de desenvolvimento:
Este trabalho é de caráter estritamente individualmente. Por favor não tente emprestar sua implementação para outros colegas, nem copiar deles, pois isso invalida a avaliação de todos os envolvidos.

Ferramentas de desenvolvimento:
0 programa deve ser compilável no Dev-C++ versão 4.9.9.2. sob Windows XP ou Vista.

O que entregar:
Apenas o arquivo *trabalho.cpp* extraído do projeto implementado, observando que ele deve conter todas as rotinas necessárias para a execução das funções solicitadas.

Como entregar:
A entrega será realizada via sistema COL até a data e hora de inicio da P3. EPs entregues após este horário serão desconsiderados.

Critérios de avaliação:

- Função *manutencao*: até 3,0 pontos.
- Função *sucessor*: até 4,0 pontos (necessita que a função *manutencao* esteja 100% correta).
- Função *folhaMaisProxima*. até 3,0 pontos (idem).

Tendo em vista a necessidade de garantir a correção das funções mais simples para testar as mais complexas, sugere-se fortemente que estas sejam implementadas na ordem acima.

O número de testes realizados na avaliação é necessariamente pequeno, e um programa que esteja funcionando apenas de forma parcial pode facilmente ficar com uma nota muito abaixo do esperado. Por este motivo, sugere-se que o programa seja testado de forma *exaustiva*, com vários tipos de entradas e com especial atenção a casos excepcionais, para assim garantir um resultado satisfatório.

Penalidade prevista:
Semelhança com outros trabalhos: nota zero a todos os envolvidos (inclusive entre turmas).

Lembrete:
A nota deste EP *não é passível de substituição*, e a avaliação será calculado como o *mínimo* entre EP e a prova correspondente.