O objetivo deste laboratório consiste em implementar o método **SecondMinimum()** que encontra o ***segundo menor valor*** em uma árvore binária de busca. Sua implementação não deve utilizar nenhum método auxiliar nem alterar a árvore. Caso não exista um segundo menor valor, uma mensagem de erro apropriada deve ser fornecida pelo método.

Implementação

Você deve implementar o método conforme o enunciado fornecido anteriormente. Utilize apenas programação estruturada. Juntamente com este documento, estão sendo disponibilizados os seguintes arquivos:

- *BSTreeTemplate.h* (interface e implementação da classe *BynarySearchTree* usando *templates*)
- *main.cpp* (programa principal para teste do método implementado)
- *StudentEmptyTest.h* (arquivo de teste do aluno, no formato da plataforma CxxTest[1])

Submissão

Submeta sua implementação no sistema Web-CAT, disponível em http://kode.ffclrp.usp.br:8080/WebCat. Este laboratório deve ser submetido individualmente.

Compacte os seguintes arquivos em um único arquivo .zip (não utilize espaços no nome do arquivo compactado, nem adicione pastas/diretórios no arquivo compactado):

- *BSTreeTemplate.h* (interface e implementação da classe *BynarySearchTree* usando *templates*)
- *StudentEmptyTest.h* (arquivo de teste do aluno, no formato da plataforma CxxTest)

Não inclua o programa principal, ou seja a função *main*(), na submissão ao Web-CAT. Respeite os nomes de arquivos, da classe e dos métodos. Submeta o arquivo compactado ao Web-CAT. Em caso de dúvida, procure o professor.

[1] Você pode inserir mais casos de teste neste arquivo, caso queira tenha interesse em testar com mais detalhes seu código. Consulte http://cxxtest.com/guide.html#testAssertions.

```
int BinarySearchTree::SecondMinimum()
{ TreePointer pai,t=root;

  if(t == NULL)  // arvore vazia?
  { cout << "Nao existe segundo minimo" << endl;
    abort();
  }
  if(t->LeftNode == NULL && t->RightNode == NULL) // arvore com um unico no'?
  { cout << "Nao existe segundo minimo" << endl;
    abort();
  }
  // ABB com pelo menos 2 nos
  // 1o. minimo no elemento mais a esquerda
  // 2o. min: se ha subarvore direita do 1o. min entao o 2o min
  //          encontra-se no elemento mais a esquerda da subarvore direita
  //          caso contrario, o 2o. min encontra-se no pai do 1o. min

  while (t->LeftNode != NULL) // 1o. min
  { pai = t;
    t = t->LeftNode;
  }
  // Neste ponto, t aponta para 1o. min (pai e' pai de t)
  if(t->RightNode != NULL) // t tem subarvore direita, procurar nela
  { t = t->RightNode;
    // procurar 2o min na subarvore mais a esquerda
    while(t->LeftNode != NULL)
      t = t->LeftNode;
    return t->Entry;
  }
  else // t nao tem subarvore direita, o 2o. min eh o pai
    return pai->Entry;
}
```