Estruturas de Dados e Introdução à Projeto e Análise de Algoritmos
Roteiro de Laboratório – Recursão e Árvores

1 Introdução

O conceito de *recursão* é fundamental na matemática e na ciência da computação. Uma definição simples de recursão (para ambas as áreas) usa o conceito de função recursiva, isto é, uma função que chama ela mesma (ou é definida em termos dela mesma). Também é necessário um *caso base* ou *condição de terminação* para que a recursão termine. Apesar de recursão ser um conceito simples, todo o paradigma de programação funcional se baseia nele. De fato, conforme mostrado por Alonso Church na década de 1930 com o formalismo de λ-*calculus*, qualquer computação pode ser resumida a definições de funções e suas aplicações (possivelmente recursivas).

O estudo de recursão aqui é interlaçado com o estudo de estruturas recursivamente definidas conhecidas como *árvores*. Vamos usar árvores tanto como estruturas de dados explícitas, bem como construções auxiliares para a compreensão e análise de programas recursivos. Recursão nos ajuda a desenvolver estruturas de dados e algoritmos elegantes e eficientes para inúmeras aplicações.

O objetivo deste laboratório é relembrar os conceitos de árvores e de recursão vistos na disciplina de Estrutura de Dados, expandindo o uso desses conceitos para algoritmos não-recursivos.

2 *Binary Search Tree (ABB - Árvore de Busca Binária - ABB)*

Uma **árvore binária** (*binary tree*) é formada por um *nó externo (folha)* ou por um *nó interno* conectado a um par de árvores binárias, chamadas de sub-árvores da esquerda e da direita do nó, respectivamente. Uma **árvore binária de busca** (*binary search tree – ABB*) é uma árvore binária aonde cada nó possui uma *chave*, e satisfaz a seguinte propriedade adicional: para qualquer nó n da árvore, a chave de n é (i) maior do que as chaves de todos os nós da sub-árvore da esquerda de n; e (ii) menor do que as chaves de todos os nós da sub-árvore da direita de n. Por essa definição é imediato notar que uma ABB não admite chaves repetidas.

Exercício 1 – TAD ABB. Crie um TAD para uma ABB assumindo que as chaves são valores inteiros (tipo `int`). Note que foi pedido um TAD, o que implica que o nó da ABB deve ser uma estrutura opaca (isto é, com a implementação "escondida" do cliente). A seguir, implemente as seguintes operações:

1. Criação de uma ABB (com um elemento).
2. Inserção de um elemento na ABB (recursivo e iterativo).

A inserção de uma chave que já existe na ABB termina silenciosamente sem modificar a estrutura (verificar se isso realmente acontece).

Exercício 2 – Cliente do TAD ABB. Uma ABB é uma estrutura não balanceada. Uma sequência de inserção de chaves infeliz (ordenada) leva a uma ABB *degenerada*, isto é, uma árvore aonde todas as sub-árvores da esquerda (ou da direita) são vazias. Neste caso a árvore "degenera" para uma lista encadeada.

Uma forma de se tentar evitar a construção de uma ABB desbalanceada é causar alguma "pertubação" na sequência de inserções na tentativa de evitar que todas as chaves fiquem de um único lado. Uma "pertubação" normalmente é gerada por algum efeito aleatório, seja através de trocas aleatórias das posições de chaves na sequência de inserção, seja através da simples geração de chaves aleatórias.

Implemente um programa cliente (ele é fornecido junto com este roteiro) para a implementação da ABB que:

- Recebe como argumento um número N de chaves a serem geradas.
- Cria uma ABB.
- Gera N chaves (`int`s) aleatórias e insere na ABB.
- Determina a altura final da ABB criada e exibe essa informação no terminal. (Veja, por exemplo, os exercícios de caminhamento em árvore a seguir.) Lembrando que a altura de uma árvore é o comprimento do caminho mais longo da raiz até alguma folha. A altura de uma árvore com um único nó raiz é *zero* e de uma árvore vazia é -1.

Para esses exercícios é suficiente utilizarmos números pseudo-aleatórios. A forma mais simples de fazer isso em Python é com o código abaixo.

```
from random import randint
print(randint(l_inf,l_sup))
```

Veja `https://en.wikipedia.org/wiki/Pseudorandomness` para mais detalhes sobre o surgimento e uso de geradores de números pseudo-aleatórios.

Concluído o seu programa cliente, realize experimentos para verificar a inserção de elementos na árvore criada, verificando sempre sua altura e impressão.

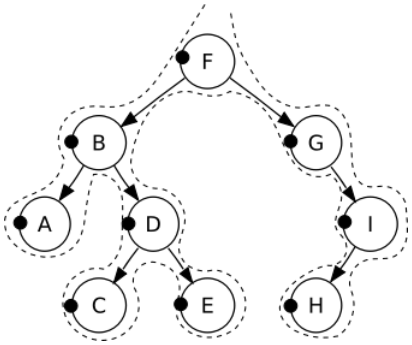
3 Caminhamento em árvore (*tree traversal*)

Um caminhamento em profundidade em uma árvore binária pode ser realizado de três formas distintas. Em todos os casos, todos os nós da árvore são *visitados*, representando uma abstração da operação que se quer realizar sobre cada nó durante o caminhamento. Um exemplo trivial de uma operação de visitação é exibir o conteúdo do nó. Os três caminhamentos em profundidade diferem somente do momento quando um nó é visitado. Todas as três formas podem ser facilmente descritas através de algoritmos recursivos.

Recursive pre-order traversal. Dado um nó t, o caminhamento em pré-ordem de t:

1. Visita t.
2. Caminha recursivamente em pré-ordem na sub-árvore da esquerda.
3. Caminha recursivamente em pré-ordem na sub-árvore da direita.

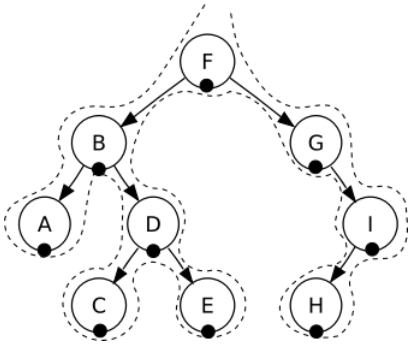
A figura ao lado exibe um caminhamento em pré-ordem em uma ABB. Os nós são visitados na ordem: F, B, A, D, C, E, G, I, H.



Recursive in-order traversal. Dado um nó t, o caminhamento em ordem de t:

1. Caminha recursivamente em ordem na sub-árvore da esquerda.
2. Visita t.
3. Caminha recursivamente em ordem na sub-árvore da direita.

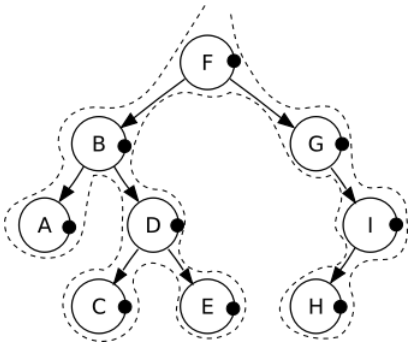
A figura ao lado exibe um caminhamento em ordem em uma ABB. Os nós são visitados na ordem: A, B, C, D, E, F, G, H, I. Como o nome já diz, um caminhamento em ordem em uma ABB recupera as chaves de forma ordenada.



Recursive post-order traversal. Dado um nó t, o caminhamento em pós-ordem de t:

1. Caminha recursivamente em pós-ordem na sub-árvore da esquerda.
2. Caminha recursivamente em pós-ordem na sub-árvore da direita.
3. Visita t.

A figura ao lado exibe um caminhamento em pós-ordem em uma ABB. Os nós são visitados na ordem: A, C, E, D, B, H, I, G, F.



Exercício 3 – ***Recursive ABB traversal.*** Implemente as três formas de caminhamento em profundidade descritas acima através de funções recursivas.

4 Caminhamento não-recursivo

É importante considerarmos também implementações não-recursivas dos algoritmos de caminhamento. As três variantes dos caminhamentos em profundidade podem ser implementadas utilizando uma pilha de nós da ABB.

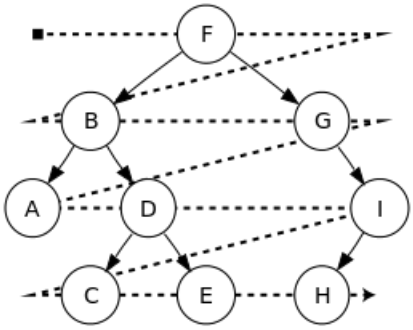
Exercício 4 – TAD Pilha. Implemente um TAD pilha que permite o empilhamento (`push`) e remoção (`pop`) de nós da ABB. A sua implementação pode ser bastante simples. É aceitável uma implementação com um limite máximo para a pilha. O foco aqui é o uso das operações de pilha no caminhamento, não uma implementação avançada dessa estrutura.

Exercício 5 – *Non-recursive ABB traversal.* Utilizando a pilha desenvolvida no exercício anterior, implemente versões não-recursivas dos três algoritmos de caminhamento em profundidade: *pre-order*, *in-order* e *post-order*. Comece com o *pre-order* que é mais simples. Os outros dois são um pouco mais elaborados, mas não muito. Se estiver em dúvida, veja um exemplo de pseudo-código em `https://en.wikipedia.org/wiki/Tree_traversal`.

5 *Level-order traversal*

Árvores também podem ser exploradas em níveis (*level-order*), aonde todos os nós de um mesmo nível são visitados antes de se passar para o nível seguinte. Esse método também é chamado de caminhamento em *largura*.

A figura ao lado exibe um caminhamento em níveis em uma ABB. Os nós são visitados na ordem: F, B, G, A, D, I, C, E, H.



Curiosamente, é razoavelmente difícil construir um algoritmo recursivo de caminhamento por níveis sem usar uma estrutura auxiliar. No caso de um algoritmo iterativo, basta utilizar uma fila simples de nós.

Exercício 6 – TAD Fila. Implemente um TAD fila que permite o enfileiramento (`enqueue`) e remoção (`dequeue`) de nós da ABB. Valem os mesmos comentários do exercício 4.

Exercício 7 – *Non-recursive level-order traversal.* Utilizando a fila desenvolvida no exercício anterior, implemente o algoritmo não-recursivo de caminhamento em níveis.

Observações – *Alterações nos TADs fornecidos são permitidas para as funções já implementas.*