MAC0323 – Algoritmos e Estruturas de Dados II

Bacharelado em Ciência da Computação

Primeiro Semestre de 2024

Prova 1 – 7/5/2024

Nome completo: _____

NUSP: _____

Assinatura: _____

Instruções:

1. Não destaque as folhas deste caderno.
2. Preencha o cabeçalho acima.
3. A prova pode ser feita a lápis. Cuidado com a legibilidade.
4. A prova consta de **3 questões**. Verifique antes de começar a prova se o seu caderno de questões está completo.
5. Não é permitido o uso de folhas avulsas para rascunho.
6. Não é permitido o uso de artefatos eletrônicos.
7. Não é permitida a consulta a livros, apontamentos ou colegas.
8. Não é necessário apagar rascunhos no caderno de questões.

DURAÇÃO DA PROVA: 2 horas

Questão	Nota
1	
2	
3	
Total	

Q1 (3.0 pontos) Diga qual será a saída do programa abaixo quando executado com os dígitos de seu número USP, dados um a um, separado por espaços, seguido de `2 0 2 4`. Por exemplo, se seu NUSP fosse 31415926, você teria de simular o programa com entrada

```
3 1 4 1 5 9 2 6 2 0 2 4
```

Importante: seu rascunho deve dar alguma indicação de como você chegou a sua resposta.

Importante: seu número USP não é 31415926.

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.StdIn;

public class Q1
{
    // Sort array a making some number of passes.
    // In each pass, pairs of consecutive non-decreasing
    // runs are merged.
    public static void sort(int[] a) {
        int[] aux = new int[a.length];
        while (true) {
            int runs = 0;  // will count number of runs as we go along
            int s = 0;     // start of first run
            while (s < a.length) {
                int t = nextStart(a, s);
                runs++;
                if (t == a.length)
                    break;
                int tt = nextStart(a, t);
                merge(a, aux, s, t, tt);
                s = tt;
            }
            StdOut.println(runs + " run(s):");
            show(a);
            if (runs == 1) // now have only one run: array is sorted
                break;
        }
    }

    // finds the start of the next non-decreasing run
    private static int nextStart(int[] a, int s) {
        while (s + 1 < a.length && a[s] <= a[s + 1])
            s++;
        return s + 1;
    }

    private static void show(int[] a) {
        for (int i = 0; i < a.length; i++)
            StdOut.print(a[i] + " ");
        StdOut.println();
    }
```

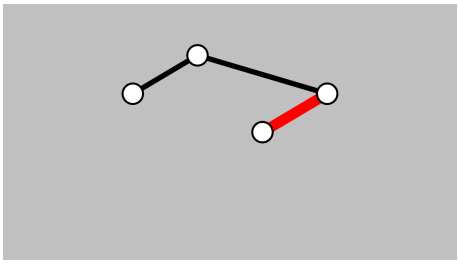
```
    // stably merge a[lo .. mid) with a[mid .. hi) using aux[lo .. hi)
    private static void merge(int[] a, int[] aux, int lo, int mid, int hi) {

        for (int k = lo; k < hi; k++)
            aux[k] = a[k];

        int i = lo, j = mid;
        for (int k = lo; k < hi; k++) {
            if      (i >= mid)              a[k] = aux[j++];
            else if (j >= hi)               a[k] = aux[i++];
            else if (aux[j] < aux[i])       a[k] = aux[j++];
            else                            a[k] = aux[i++];
        }
    }

    public static void main(String[] args)
    {
        int[] a = StdIn.readAllInts();
        show(a);
        sort(a);
        show(a);
    }
}
```

Rascunho

Rascunho

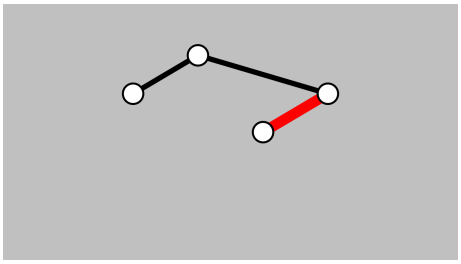
Saída do programa

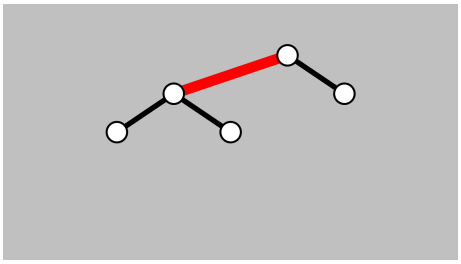
Q2 (3.0 pontos) Nesta questão, tratamos de árvores rubro-negras esquerdistas (ARNEs). Supomos que as chaves em nossas ARNEs são cadeias de caracteres (strings). Suponha que inserimos as chaves BB, AA, DD, CC em uma ARNE inicialmente vazia, nessa ordem. Obtemos uma ARNE com o seguinte formato ao final do processo:

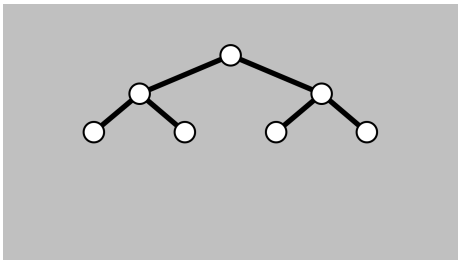


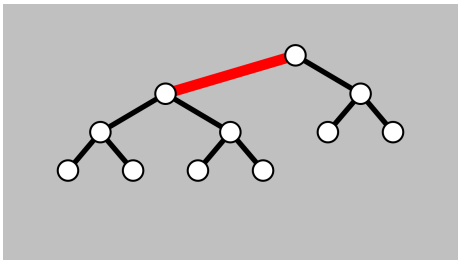
(Na ARNE acima, há exatamente um nó/link vermelho.)

Nos itens abaixo, dê uma sequência ordenada de chaves (strings) que resulta na ARNE dada. Por exemplo, para (a), uma possível resposta seria BB, AA, DD, CC `:-)` Não deixe de responder o item (a)!

(a)  Resposta:

(b)  Resposta:

(c)  Resposta:

(d)  Resposta:

Q3 (4.0 pontos) Esta questão considera árvores binárias de busca (ABBs). Esta questão também supõe que você lembra como ABBs são implementadas no programa `BST.java`. Em particular, lembre que os nós da ABB são implementadas como objetos do tipo `Node`:

```
private class Node {
    private Key key;           // sorted by key
    private Value val;         // associated data
    private Node left, right;  // left and right subtrees
    private int size;          // number of nodes in subtree

    public Node(Key key, Value val, int size) {
        this.key = key;
        this.val = val;
        this.size = size;
    }
}
```

Vamos dizer aqui que uma árvore binária T é *perfeitamente balanceada* se, para cada nó x de T, vale que o número de nós na subárvore esquerda de x é $\lfloor x.size/2 \rfloor$ (lembre que $x.size$ é o número de nós na subárvore enraizada em x).

A AB vazia é perfeitamente balanceada e a AB com um único nó é também perfeitamente balanceada. Para cada N, existe exatamente uma AB perfeitamente balanceada.

(a) Desenhe as ABs perfeitamente balanceadas com N nós, para cada $2 \leq N \leq 9$.

(b) Suponha que um cliente de `BST.java` quer verificar se a ABB atual é perfeitamente balanceada. Isto é, o cliente faz algo como

```
BST<String, Integer> st = new BST<String, Integer>();
```

e mais à frente ele quer fazer

```
StdOut.println("BST currently perfectly balanced: " + st.isPB());
```

para saber se a ABB está perfeitamente balanceada. Implemente a função `isPB()`: isto é, implemente uma função de protótipo

```
public boolean isPB()
```

que pode ser usada como acima, supondo que sua função será adicionada ao arquivo `BST.java`.

Considere as rotinas de rotação à esquerda e rotação à direita para ABBs dadas a seguir:

```
private Node rotateLeft(Node h) {
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.size = h.size;
    h.size = size(h.left) + size(h.right) + 1;
    return x;
}
```

```
private Node rotateRight(Node h) {
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.size = h.size;
    h.size = size(h.left) + size(h.right) + 1;
    return x;
}
```

Lembre também que sabemos como encontrar a chave de posto k em uma ABB (isto é, a chave x tal que há exatamente k chaves menos que x na ABB): basta executar `select(k)`, onde o método `select()` é como dado a seguir.

```
public Key select(int rank) {
    return select(root, rank);
}

// Returns key of the given rank in the BST rooted at x
private Key select(Node x, int rank) {
    if (x == null) return null;
    int leftSize = size(x.left);
    if      (leftSize > rank) return select(x.left,  rank);
    else if (leftSize < rank) return select(x.right, rank - leftSize - 1);
    else                      return x.key;
}
```

Queremos adicionar uma rotina chamada `balance()` a `BST.java` que transforma nossa ABB na ABB perfeitamente balanceada correspondente, quando executamos `st.balance()` (supondo que `st` é nosso objeto do tipo `BST`). O seguinte código implementa `balance()` (supondo a existência de uma certa função `partition()`):

```
public void balance() {
    root = balance(root);
}

private Node balance(Node x) {
    if (x == null) return null;
    if (x.size < 2) return x;
    x = partition(x, x.size/2);
    x.left = balance(x.left);
    x.right = balance(x.right);
    return x;
}
```

A chamada `partition(x, k)` deve reorganizar os nós da ABB T_x enraizada em `x` de forma que a nova raiz seja o nó que contém a chave de posto `k` em T_x. Além disso, `partition(x, k)` deve devolver esta nova ABB. Note que, com tal função `partition()`, a função `balance()` acima de fato transforma nossa ABB em uma ABB perfeitamente balanceada.

(c) Implemente aqui a função `partition()`, como descrita acima. Sua função `partition()` deve ter assinatura

```
private Node partition(Node x, int rank)
```

Sugestão. Releia a função `select()`. Basta fazer adaptações no código de `select()`, inserindo certas rotações, para trazer o nó desejado para a raiz da árvore.

* * *

(Rascunho)