```
Daniel Bastos <dbastos@id.uff.br>
Trabalho EDA, T1-20171, data 16.06.2017.

/*

    Caro Prof. André, eis um relatório justificando as escolhas das
    estruturas de dados.  O código foi escrito em Java num projeto
    NETBEANS.  Ao fim deste relatório há um exemplo de execução do
    programa.

    Para leitura do código, comece pela classe InicieLeituraAqui que
    possui o método main().

    public static void main(String[] args) {
        InicieLeituraAqui x = new InicieLeituraAqui();
        x.perguntaTipo1();
        x.perguntaTipo2();
        x.perguntaTipo3();
    }

    As estruturas de dados que implementei foram MyLinkedList
    (MyLinkedList.java), HashTable (HashTable.java) e AVLTree
    (AVLTree.java).

    O construtor de InicieLeituraAqui inicializa cada uma das
    estruturas de dados usadas de forma que quando uma pergunta tipo
    1, 2 ou 3 seja feita, todas as estruturas estão prontas para serem
    consultadas.

    [...]

    // Tabela hash para responder pergunta tipo 1.
    HashTable filiais = new HashTable();

    // Tabela hash (equipada com árvore AVL) para perguntas tipo 2.
    HashTable filiaisPeriodo = new HashTable();

    // Lista para otimizar a pergunta tipo 3.
    MyLinkedList todasFiliais = new MyLinkedList();

    public InicieLeituraAqui() {
        makeDb();
        makeHashFiliais();
        makeHashWithAVL();
    }

    [...]

    (*) Pergunta tipo 1

    Na pergunta tipo 1 é importante encontrar rapidamente o total de
    uma determina filial.  Pra isso, construí uma tabela hash que
    armazena em cada /bucket/ o total de uma determinada filial.  Se o
    usuário deseja o total de filiais de um intervalo [x, y] de
    filiais, o tempo de resposta é Theta(n), onde n = |y - x|.  A
    essência do algoritmo é descrito pelo método
    totalVendasFiliais(int x, int y).

    public void perguntaTipo1() {
        System.out.println("Pergunta tipo 1.\n");
        System.out.format("Total filiais %d a %d: R$ %.2f\n",
                10, 20, totalVendasFiliais(10, 20));
        System.out.println();
    }

    public double totalVendasFiliais(int filial_i, int filial_j) {
        double total = 0;
        for (int i = filial_i; i <= filial_j; ++i) {
            Object x = filiais.get(i);
```

```
        if (x != null) {
            total = total + (double) x;
        }
    }
    return total;
}

A construção da tabela hash é feita pela função makeHashFiliais().

private void makeHashFiliais() {
    // Constrói tabela hash contendo o total de vendas de uma
    // filial.  Com essa tabela hash, obtemos o total de uma
    // filial em tempo constante.

    for (int i = 0; i < db.getSize(); ++i) {
        Venda v = (Venda) db.get(i);
        Object x = filiais.get(v.filial);
        if (x == null) {
            filiais.set(v.filial, v.total_vendido);
        } else {
            filiais.set(v.filial, (double) x + v.total_vendido);
        }
    }
}

(*) Pergunta tipo 2

Na pergunta tipo 2, usei uma tabela hash para encontrar os dados
de uma determinada filial.  Em cada /bucket/ dessa tabela hash,
armazeno um conjunto cujos elementos representam o total de vendas
daquela filial num determinado ano/mês.  O conjunto é implementado
usando uma árvore AVL.

Para responder à pergunta tipo 2, o algoritmo varre o período de
filiais desejado pelo usuário fazendo a interseção entre o periodo
de meses.  Para obter cada F especificamente, efetuamos um get()
na tabela hash gastando tempo Theta(1) em média.  Para obter cada
mês, tempo Theta(lg m) é gasto onde m é o números de meses
inseridos na árvore AVL.

O algoritmo tem complexidade

    O(f * m lg m)

sendo f = |f2 - f1|, o número de filials, m = |d2 - d1|, o número
de meses contido no período.

public void perguntaTipo2() {
    String d1 = null;
    String d2 = null;
    int f = 0;
    int t = 0;

    System.out.println("Pergunta tipo 2.\n");

    f = t = 18; d1 = "2017-07"; d2 = "2017-08";
    System.out.format("Total filiais %d a %d, período %s a %s: R$ %.2f\n",
            f, f, d1, d2, totalFiliaisPeriodoIntersecao(f, t, d1, d2));

    f = 18; t = 21; d1 = "2017-01"; d2 = "2018-12";
    System.out.format("Total filiais %d a %d, período %s a %s: R$ %.2f\n",
            f, t, d1, d2, totalFiliaisPeriodoIntersecao(f, t, d1, d2));

    f = 1; t = 100; d1 = "2017-10"; d2 = "2017-11";
    System.out.format("Total filiais %d a %d, período %s a %s: R$ %.2f\n",
            f, t, d1, d2, totalFiliaisPeriodoIntersecao(f, t, d1, d2));

    System.out.println();
}
```

```
public double totalFiliaisPeriodoIntersecao(int fi, int fj,
        String d1, String d2) {
    double ret = 0;
    for (int fx = fi; fx <= fj; ++fx) {
        for (DateTime d = new DateTime(d1);
                d.isBefore(new DateTime(d2).plusSeconds(1));
                d = d.plusMonths(1)) {
            Filial f = (Filial) filiaisPeriodo.get(fx);
            if (f != null) {
                Object x = f.tree.get(getKeyFromDate(d));
                if (x != null) {
                    Montante m = (Montante) x;
                    ret += m.total;
                }
            }
        }
    }
    return ret;
}
```

A construção da árvore é feita pelo método makeHashWithAVL().

```
private void makeHashWithAVL() {
    for (int i = 0; i < db.getSize(); ++i) {
        Venda v = (Venda) db.get(i);

        Filial x = (Filial) filiaisPeriodo.get(v.filial);
        if (x == null) {
            x = new Filial(v.filial);
            filiaisPeriodo.set(v.filial, x);
            todasFiliais.add(v.filial); // para pergunta tipo 3
        }

        x.tree.insert(new Montante(v.ano_mes, v.total_vendido));
    }
}
```

(*) Pergunta tipo 3

Na pergunta tipo 3 já sabemos que queremos todas as filiais. Por isso, mantive uma lista encadeada contendo o id de todas as filiais. Assim o algoritmo para responder a esse tipo de pergunta é apenas varrer a lista de todas as filiais, usando a pergunta tipo 2 para fazer a interseção de conjuntos quanto ao /período/ desejado pelo usuário.

```
public double totalTodasFiliais(String d1, String d2) {
    double total = 0.0;
    for (int i = 0; i < todasFiliais.getSize(); ++i) {
        int filial = (int) todasFiliais.get(i);
        total += totalFiliaisPeriodoIntersecao(filial, filial, d1, d2);
    }
    return total;
}
```

A complexidade é O(n^2 lg(m)), onde n é o número de filiais e m = |d2 - d1|, o número meses no intervalo.

(*) Sobre a implementação da tabela hash

Knuth [em The Art of Computer Programming, volume 3, capítulo 6.4 ``HASHING'', página 516] afirma que o número primo 1009 ``has been found to be quite satisfactory in most cases.'' Para resolver colisões, usei uma classe privada chamada HashItem que é uma lista encadeada --- veja HashTable.java.

(*) Exemplo de execução do programa

```
   run:
   Pergunta tipo 1.

   Total filiais 10 a 20: R$ 43406,10

   Pergunta tipo 2.

   Total filiais 18 a 18, período 2017-07 a 2017-08: R$ 5138,80
   Total filiais 18 a 21, período 2017-01 a 2018-12: R$ 60444,20
   Total filiais 1 a 100, período 2017-10 a 2017-11: R$ 25694,00

   Pergunta tipo 3.

   Total todas filiais, período 2017-05 a 2017-09: R$ 34750,20
   Total todas filiais, período 2000-01 a 2020-12: R$ 60444,20

*/
```