Programa de Pós-Graduação em Computação – IC/UFF
Relatório do Trabalho Prático de Estrutura de Dados e Algoritmos – 2017.2
Docente: André Luiz Portes Paes Leme
Discentes: Mayara Tesch Carreiro e Fabiano da Guia Rocha

Estruturas utilizadas e análise de complexidade:
Tabela Hash: estrutura utilizada para o acúmulo dos fluxos de mesma chave (setorData) que serão utilizados na criação/atualização da arvore AVL. A estrutura foi escolhida por ter complexidade de inserção constante O(1). A chave utilizada para tabela hash foi a junção da data(aammdd) com o setor(ss), resultando em um valor inteiro no formato (aammddss). Rodovias diferentes foram consideradas quando há a repetição da chave setorData.

Árvore AVL principal: estrutura de dados criada tendo como chave (fluxo) armazenando o valor setorData em seu atributo em uma lista encadeada. A estrutura foi escolhida por ter complexidade O(log n).

Árvore AVL de atualização: com o objetivo de obter melhor desempenho na atualização da Árvore AVL principal por não precisar percorrer em O(n) a Tabela Hash. A cada novo fluxo armazenado é criado um novo nó na Arvore AVL de atualização contendo o valor da chave setorData, para posterior atualização da Árvore AVL principal na ordem O(log n).

Exemplo de Entrada (conforme classe principal – insere(aammddss,fluxo)):

Primeiro bloco de atualizações
```
hashTable.insere(17101001, 151);
hashTable.insere(17081525, 181);
hashTable.insere(17041216, 530);
hashTable.insere(17041516, 201);
hashTable.insere(17081510, 152);
hashTable.insere(17101003, 541);
hashTable.insere(17070507, 541);
hashTable.insere(17032704, 182);
hashTable.insere(17081408, 81);
hashTable.insere(17031525, 201);
hashTable.insere(17101101, 151);
hashTable.insere(17081325, 181);
hashTable.insere(17041116, 530);
hashTable.insere(17041716, 201);
hashTable.insere(17080710, 152);
hashTable.insere(17100803, 541);
hashTable.insere(17070307, 541);
hashTable.insere(17032304, 182);
hashTable.insere(17081808, 210);
hashTable.insere(17031125, 201);
```

Segundo bloco de atualizações
```
hashTable.insere(17101011, 530);
hashTable.insere(17101011, 1);
hashTable.insere(17081515, 521);
hashTable.insere(17041206, 495);
hashTable.insere(17041506, 500);
hashTable.insere(17081511, 652);
hashTable.insere(17101004, 341);
hashTable.insere(17070509, 641);
hashTable.insere(17032711, 382);
hashTable.insere(17081418, 381);
hashTable.insere(17031505, 401);
hashTable.insere(17101111, 351);
hashTable.insere(17081313, 10);
hashTable.insere(17041118, 30);
hashTable.insere(17041712, 501);
hashTable.insere(17080711, 352);
hashTable.insere(17100824, 241);
hashTable.insere(17070317, 341);
hashTable.insere(17032302, 582);
hashTable.insere(17081801, 410);
hashTable.insere(17031121, 73);
```

Exemplo de Saída(conforme dados da classe principal):

Primeiro bloco de atualizações
```
Fluxo Mínimo: 81
Fluxo Máximo: 541
Limiar: 449.0
Saída: {setor,dia,fluxo}
{16,170411,530}
{16,170412,530}
{7,170703,541}
{7,170705,541}
{3,171008,541}
{3,171010,541}
```

Segundo bloco de atualizações
```
Fluxo Mínimo: 10
Fluxo Máximo: 652
Limiar: 523.6
Saída: {setor,dia,fluxo}
{16,170411,530}
{16,170412,530}
{11,171010,531}
{7,170703,541}
{7,170705,541}
{3,171008,541}
{3,171010,541}
{2,170323,582}
{9,170705,641}
{11,170815,652}
```

Classe Hash

Descrição: classe que contém métodos responsáveis por receber os valores de entrada e armazenar em uma tabela hash através de um atributo chave conhecido.

```
package trabalho_ed;

public class Hash {
    private final int numeroEsperadoDeElementos;
```

```
    private ListaHash[] listahash;
    private ArvoreAvl updateTree = new ArvoreAvl();

    public Hash(int numEsperado) {
        numeroEsperadoDeElementos = numEsperado;
        listahash = new ListaHash[numEsperado];
        for (int i = 0; i < numEsperado; i++) {
            listahash[i] = new ListaHash();
        }
    }

    public void insere(int setorData, int fluxo) {
        int indice = hash(setorData);
        listahash[indice].inserir(setorData, fluxo);
        updateTree.inserir(setorData, 0);
    }

    public DadosHash busca(int setorData) {
        int indice = hash(setorData);
        ListaHash temp = listahash[indice];
        return (temp.busca(setorData));
    }

    public boolean remove(int setorData) {
        DadosHash res = busca(setorData);
        if (res == null) {
            return false;
        }
        int indice = hash(setorData);
        listahash[indice].remove(setorData);
        return true;
    }

    public int hash(int setorData) {
        return (setorData % numeroEsperadoDeElementos);
    }
    public ListaHash[] getListaHash() {
        return this.listahash;
    }
    public ArvoreAvl getUpdateTree() {
        return this.updateTree;
    }
    public void removeUpdateTree(){
        updateTree.raiz = null;
    }
}
```

Classe ListaHash

Descrição: classe responsável por tratar as colisões que ocorrem na inserção de dados na tabela hash criando uma lista para armazenar os atributos que possuem mesmo valor de hash.

```
package trabalho_ed;

public class ListaHash {
    private DadosHash primeiro;
    private int tamanho = 0;

    public void inserir(int setorData, int fluxoAtual) {
        DadosHash n = new DadosHash(setorData, fluxoAtual);
        inserir(this.primeiro, n);
    }

    private void inserir(DadosHash aComparar, DadosHash aInserir) {
        if (aComparar == null) {
            this.primeiro = aInserir;
            tamanho++;
        } else {
            while (aComparar.getSetorData() != aInserir.getSetorData()) {
                if (aComparar.getProximo() != null) {
                    aComparar = aComparar.getProximo();
```

```
                } else if (aComparar.getSetorData() != aInserir.getSetorData()) {
                    aComparar.setProximo(aInserir);
                    tamanho++;
                    return;
                }
            }
            if (aComparar.getSetorData() == aInserir.getSetorData()) {
                aComparar.atualizarFluxoAtual(aInserir.getFluxoAtual());
            }
        }
    }

    public boolean contem(int setorData) {
        DadosHash anterior = this.primeiro;
        DadosHash atual = anterior.getProximo();

        while (anterior != null) {
            if (this.primeiro.getSetorData() == setorData) {
                this.primeiro = this.primeiro.getProximo();
                return true;
            } else if (atual.getSetorData() == setorData) {
                anterior.setProximo(atual.getProximo());
                return true;
            }
            anterior = anterior.getProximo();
            atual = atual.getProximo();
        }
        return false;
    }

    public void imprimeLista() {
        DadosHash aux = this.primeiro;
        if (aux == null) {
            System.out.println("nulo");
        }
        while (aux != null) {
            System.out.println("Anterior" + aux.getFluxoAntigo() + " Atual:" +
aux.getFluxoAtual());
            aux = aux.getProximo();
        }
    }

    public DadosHash busca(int setorData) {
        DadosHash temp = primeiro;
        while (temp != null) {
            if (temp.getSetorData() == setorData) {
                return temp;
            } else {
                temp = temp.getProximo();
            }
        }
        return null;
    }

    public boolean remove(int setorData) {
        DadosHash temp = primeiro;
        if (primeiro.getSetorData() == setorData) {
            primeiro = primeiro.getProximo();
            return true;
        } else {
            while (temp.getProximo() != null) {
                if (temp.getProximo().getSetorData() == setorData) {
                  temp.getProximo().setProximo(temp.getProximo().getProximo());
                  return true;
                }
                temp = temp.getProximo();
            }
            return false;
        }
    }
```

```
    public int getTamanho() {
        return tamanho;
    }
    public DadosHash getPrimeiro() {
        return this.primeiro;
    }
}
```

Classe DadosHash

Descrição: classe utilizada para definir os dados que serão armazenados na tabela Hash sendo setorData e fluxo (atual e antigo), que são necessários para atualizar a arvore AVL.

```
package trabalho_ed;

public class DadosHash {
    private DadosHash proximo;
    private int setorData;
    private int fluxoAtual;
    private int fluxoAntigo;

    public DadosHash(int setorData, int fluxoAtual) {
        this.setorData = setorData;
        this.fluxoAtual = fluxoAtual;
        this.fluxoAntigo = -1;
    }

    public void setProximo(DadosHash proximo) {
        this.proximo = proximo;
    }
    public DadosHash getProximo() {
        return proximo;
    }
    public int getFluxoAtual() {
        return fluxoAtual;
    }
    public void atualizarFluxoAtual(int valor) {
        fluxoAtual += valor;
    }
    public int getFluxoAntigo() {
        return fluxoAntigo;
    }
    public void atualizarFluxoAntigo() {
        fluxoAntigo = fluxoAtual;
    }
    public int getSetorData() {
        return setorData;
    }
}
```

Classe ArvoreAVL

Descrição: classe responsável pelos métodos de manipulação dos nós da árvore AVL.

```
package trabalho_ed;

public class ArvoreAvl {
    protected NoAvl raiz;

    public void inserir(int chave, int valor) {
        NoAvl n = new NoAvl(chave);
        inserirAVL(this.raiz, n, valor);
    }

    private void inserirAVL(NoAvl aComparar, NoAvl aInserir, int valor) {
        if (aComparar == null) {
            this.raiz = aInserir;
            aInserir.inserirNaLista(valor);
        } else {
            if (aInserir.getChave() < aComparar.getChave()) {
                if (aComparar.getEsquerda() == null) {
```

```
                    aComparar.setEsquerda(aInserir);
                    aInserir.inserirNaLista(valor);
                    aInserir.setPai(aComparar);
                    verificarBalanceamento(aComparar);
                } else {
                    inserirAVL(aComparar.getEsquerda(), aInserir, valor);
                }
            } else if (aInserir.getChave() > aComparar.getChave()) {
                if (aComparar.getDireita() == null) {
                    aComparar.setDireita(aInserir);
                    aInserir.inserirNaLista(valor);
                    aInserir.setPai(aComparar);
                    verificarBalanceamento(aComparar);
                } else {
                    inserirAVL(aComparar.getDireita(), aInserir, valor);
                }
            } else {
                aComparar.inserirNaLista(valor);
            }
        }
    }

    private void verificarBalanceamento(NoAvl atual) {
        setBalanceamento(atual);
        int balanceamento = atual.getBalanceamento();
        if (balanceamento == -2) {
            if (altura(atual.getEsquerda().getEsquerda()) >=
altura(atual.getEsquerda().getDireita())) {
                atual = rotacaoDireita(atual);
            } else {
                atual = duplaRotacaoEsquerdaDireita(atual);
            }
        } else if (balanceamento == 2) {
            if (altura(atual.getDireita().getDireita()) >=
altura(atual.getDireita().getEsquerda())) {
                atual = rotacaoEsquerda(atual);
            } else {
                atual = duplaRotacaoDireitaEsquerda(atual);
            }
        }
        if (atual.getPai() != null) {
            verificarBalanceamento(atual.getPai());
        } else {
            this.raiz = atual;
        }
    }

    public void remover(int k, int setorData) {
        removerAVL(this.raiz, k, setorData);
    }

    private void removerAVL(NoAvl atual, int k, int setorData) {
        if (atual == null) {
            return;
        } else {
            if (atual.getChave() > k) {
                removerAVL(atual.getEsquerda(), k, setorData);
            } else if (atual.getChave() < k) {
                removerAVL(atual.getDireita(), k, setorData);
            } else if (atual.getChave() == k) {
                removerNoEncontrado(atual, setorData);
            }
        }
    }

    private void removerNoEncontrado(NoAvl aRemover, int setorData) {
        NoAvl r;
        if (aRemover.getLista().remover(setorData)) {
            if (aRemover.getLista().getPrimeiro() == null) {
                if (aRemover.getEsquerda() == null || aRemover.getDireita() == null) {
                    if (aRemover.getPai() == null) {
```

```
                    this.raiz = null;
                    aRemover = null;
                    return;
                }
                r = aRemover;
            } else {
                r = sucessor(aRemover);
                aRemover.setChave(r.getChave());
            }
            NoAvl p;
            if (r.getEsquerda() != null) {
                p = r.getEsquerda();
            } else {
                p = r.getDireita();
            }
            if (p != null) {
                p.setPai(r.getPai());
            }
            if (r.getPai() == null) {
                this.raiz = p;
            } else {
                if (r == r.getPai().getEsquerda()) {
                    r.getPai().setEsquerda(p);
                } else {
                    r.getPai().setDireita(p);
                }
                verificarBalanceamento(r.getPai());
            }
            r = null;
        }
    }
}

private NoAvl rotacaoEsquerda(NoAvl inicial) {
    NoAvl direita = inicial.getDireita();
    direita.setPai(inicial.getPai());
    inicial.setDireita(direita.getEsquerda());
    if (inicial.getDireita() != null) {
        inicial.getDireita().setPai(inicial);
    }
    direita.setEsquerda(inicial);
    inicial.setPai(direita);
    if (direita.getPai() != null) {
        if (direita.getPai().getDireita() == inicial) {
            direita.getPai().setDireita(direita);
        } else if (direita.getPai().getEsquerda() == inicial) {
            direita.getPai().setEsquerda(direita);
        }
    }
    setBalanceamento(inicial);
    setBalanceamento(direita);
    return direita;
}

private NoAvl rotacaoDireita(NoAvl inicial) {
    NoAvl esquerda = inicial.getEsquerda();
    esquerda.setPai(inicial.getPai());
    inicial.setEsquerda(esquerda.getDireita());
    if (inicial.getEsquerda() != null) {
        inicial.getEsquerda().setPai(inicial);
    }
    esquerda.setDireita(inicial);
    inicial.setPai(esquerda);
    if (esquerda.getPai() != null) {
        if (esquerda.getPai().getDireita() == inicial) {
            esquerda.getPai().setDireita(esquerda);
        } else if (esquerda.getPai().getEsquerda() == inicial) {
            esquerda.getPai().setEsquerda(esquerda);
        }
    }
    setBalanceamento(inicial);
```

```
    setBalanceamento(esquerda);
    return esquerda;
}

private NoAvl duplaRotacaoEsquerdaDireita(NoAvl inicial) {
    inicial.setEsquerda(rotacaoEsquerda(inicial.getEsquerda()));
    return rotacaoDireita(inicial);
}

private NoAvl duplaRotacaoDireitaEsquerda(NoAvl inicial) {
    inicial.setDireita(rotacaoDireita(inicial.getDireita()));
    return rotacaoEsquerda(inicial);
}

private NoAvl sucessor(NoAvl q) {
    if (q.getDireita() != null) {
        NoAvl r = q.getDireita();
        while (r.getEsquerda() != null) {
            r = r.getEsquerda();
        }
        return r;
    } else {
        NoAvl p = q.getPai();
        while (p != null && q == p.getDireita()) {
            q = p;
            p = q.getPai();
        }
        return p;
    }
}

private int altura(NoAvl atual) {
    if (atual == null) {
        return -1;
    }
    if (atual.getEsquerda() == null && atual.getDireita() == null) {
        return 0;
    } else if (atual.getEsquerda() == null) {
        return 1 + altura(atual.getDireita());
    } else if (atual.getDireita() == null) {
        return 1 + altura(atual.getEsquerda());
    } else {
        return 1 + Math.max(altura(atual.getEsquerda()), altura(atual.getDireita()));
    }
}

private void setBalanceamento(NoAvl no) {
    no.setBalanceamento(altura(no.getDireita()) - altura(no.getEsquerda()));
}

public int minFluxo() {
    return minFluxo(raiz, 0);
}

private int minFluxo(NoAvl no, int min) {
    NoAvl aux = no;
    while (aux != null) {
        min = aux.getChave();
        aux = aux.getEsquerda();
    }
    return min;
}

public int maxFluxo() {
    return maxFluxo(raiz, 0);
}

private int maxFluxo(NoAvl no, int max) {
    NoAvl aux = no;
    while (aux != null) {
        max = aux.getChave();
```

```
            aux = aux.getDireita();
        }
        return max;
    }

    public void emOrdem(double limiar) {
        ArvoreAvl.this.emOrdem(raiz, limiar);
    }

    private void emOrdem(NoAvl no, double limiar) {
        NoAvl aux = no;
        if (aux != null) {
            ArvoreAvl.this.emOrdem(aux.getEsquerda(), limiar);
            if (aux.getChave() > limiar) {
                aux.imprimeLista(aux.getChave());
            }
            ArvoreAvl.this.emOrdem(aux.getDireita(), limiar);
        }
    }

    //updateTree
    public void emOrdem(Hash hashTable, ArvoreAvl tree) {
        emOrdem(raiz, hashTable, tree);
    }

    private void emOrdem(NoAvl no, Hash hashTable, ArvoreAvl tree) {
        NoAvl aux = no;
        if (aux != null) {
            emOrdem(aux.getEsquerda(), hashTable, tree);
            tree.atualizaAVL(hashTable.busca(aux.getChave()), tree);
            emOrdem(aux.getDireita(), hashTable, tree);
        }
    }

    public void atualizaAVL(DadosHash aux, ArvoreAvl arvore) {
        arvore.remover(aux.getFluxoAntigo(), aux.getSetorData());
        arvore.inserir(aux.getFluxoAtual(), aux.getSetorData());
        aux.atualizarFluxoAntigo();
    }
}
```

Classe NoAvl

Descrição: classe utilizada para criar os nós da arvore AVL tendo como chave o valor de fluxo, no qual dentre os atributos possui uma lista que irá armazenar os valores de setorData que possuem a mesma chave.

```
package trabalho_ed;

public class NoAvl {
    private NoAvl esquerda;
    private NoAvl direita;
    private NoAvl pai;
    private ListaNoAvl lista;
    private int chave;
    private int balanceamento;

    public NoAvl(int k) {
        setEsquerda(setDireita(setPai(null)));
        setBalanceamento(0);
        setChave(k);
        lista = new ListaNoAvl();
    }

    public int getChave() {
        return chave;
    }
    public void setChave(int chave) {
        this.chave = chave;
    }
    public int getBalanceamento() {
```

```
        return balanceamento;
    }
    public void setBalanceamento(int balanceamento) {
        this.balanceamento = balanceamento;
    }
    public NoAvl getPai() {
        return pai;
    }
    public NoAvl setPai(NoAvl pai) {
        this.pai = pai;
        return pai;
    }
    public NoAvl getDireita() {
        return direita;
    }
    public NoAvl setDireita(NoAvl direita) {
        this.direita = direita;
        return direita;
    }
    public NoAvl getEsquerda() {
        return esquerda;
    }
    public void setEsquerda(NoAvl esquerda) {
        this.esquerda = esquerda;
    }
    public void inserirNaLista(int setorData) {
        lista.inserir(setorData);
    }
    public void imprimeLista(int fluxo) {
        lista.imprimeLista(fluxo);
    }
    public ListaNoAvl getLista() {
        return lista;
    }
}
```

Classe ListaNoAvl

Descrição: classe responsável por armazenar os nós que possuem a mesma chave (fluxo) a ser inserido na arvore AVL.

```
package trabalho_ed;

public class ListaNoAvl {
    private NoLista primeiro;

    public void inserir(int k) {
        NoLista n = new NoLista(k);
        inserir(this.primeiro, n);
    }

    private void inserir(NoLista aComparar, NoLista aInserir) {
        if (aComparar == null) {
            this.primeiro = aInserir;
        } else {
            while (aComparar.getProximo() != null) {
                aComparar = aComparar.getProximo();
            }
            aComparar.setProximo(aInserir);
        }
    }

    public boolean remover(int setorData) {
        NoLista anterior = this.primeiro;
        NoLista atual = anterior.getProximo();

        while (anterior != null) {
            if (this.primeiro.getValor() == setorData) {
                this.primeiro = this.primeiro.getProximo();
                return true;
            } else if (atual.getValor() == setorData) {
```

```
                anterior.setProximo(atual.getProximo());
                return true;
            }
            anterior = anterior.getProximo();
            atual = atual.getProximo();
        }
        return false;
    }

    public void imprimeLista(int fluxo) {
        NoLista aux = this.primeiro;
        if (aux == null) {
            System.out.println("nulo");
        }
        //{setor,dia,fluxo}
        while (aux != null) {
            System.out.println("{"+aux.getValor()%100 +","+aux.getValor()/100 +","+ fluxo
+ "}");
            aux = aux.getProximo();
        }
    }

    public NoLista getPrimeiro() {
        return this.primeiro;
    }
}
```

Classe NoLista

Descrição: classe que contém o atributo de cada nó de ListaNoAvl (setorData).

```
package trabalho_ed;

public class NoLista {
    private NoLista proximo;
    private int valor;

    public NoLista(NoLista proximo, int valor) {
        this.proximo = proximo;
        this.valor = valor;
    }

    public NoLista(int valor) {
        this.valor = valor;
    }
    public void setProximo(NoLista proximo) {
        this.proximo = proximo;
    }
    public NoLista getProximo() {
        return proximo;
    }
    public int getValor() {
        return valor;
    }
}
```

Classe Principal

Descrição: Classe Principal do Trabalho, onde encontram-se as chamadas principais para os métodos e exemplos de uso como inserção na tabela hash e a chamada para listagem {setor,dia,fluxo}.

```
package trabalho_ed;

public class Trabalho_ED {
    public static void main(String[] args) {
        ArvoreAvl noAvl = new ArvoreAvl();
        Hash hashTable = new Hash(1000);
        double limiar = 0;

        System.out.println("Primeiro bloco de atualizações");
```

```
        hashTable.insere(17101001, 151);
        hashTable.insere(17081525, 181);
        hashTable.insere(17041216, 530);
        hashTable.insere(17041516, 201);
        hashTable.insere(17081510, 152);
        hashTable.insere(17101003, 541);
        hashTable.insere(17070507, 541);
        hashTable.insere(17032704, 182);
        hashTable.insere(17081408, 81);
        hashTable.insere(17031525, 201);
        hashTable.insere(17101101, 151);
        hashTable.insere(17081325, 181);
        hashTable.insere(17041116, 530);
        hashTable.insere(17041716, 201);
        hashTable.insere(17080710, 152);
        hashTable.insere(17100803, 541);
        hashTable.insere(17070307, 541);
        hashTable.insere(17032304, 182);
        hashTable.insere(17081808, 210);
        hashTable.insere(17031125, 201);

        hashTable.getUpdateTree().emOrdem(hashTable, noAvl);
        hashTable.removeUpdateTree();

        System.out.println("Fluxo Minimo: " + noAvl.minFluxo());
        System.out.println("Fluxo Maximo: " + noAvl.maxFluxo());

        //min(fluxo) + 0.8Δ, onde Δ = max(fluxo) − min(fluxo).
        limiar = noAvl.minFluxo() + 0.8 * (noAvl.maxFluxo() - noAvl.minFluxo());
        System.out.println("Limiar: " + limiar);
        System.out.println("Saída: {setor,dia,fluxo}");

        noAvl.emOrdem(limiar);

        System.out.println("\nSegundo bloco de atualizações");
        hashTable.insere(17101011, 530);
        hashTable.insere(17101011, 1);
        hashTable.insere(17081515, 521);
        hashTable.insere(17041206, 495);
        hashTable.insere(17041506, 500);
        hashTable.insere(17081511, 652);
        hashTable.insere(17101004, 341);
        hashTable.insere(17070509, 641);
        hashTable.insere(17032711, 382);
        hashTable.insere(17081418, 381);
        hashTable.insere(17031505, 401);
        hashTable.insere(17101111, 351);
        hashTable.insere(17081313, 10);
        hashTable.insere(17041118, 30);
        hashTable.insere(17041712, 501);
        hashTable.insere(17080711, 352);
        hashTable.insere(17100824, 241);
        hashTable.insere(17070317, 341);
        hashTable.insere(17032302, 582);
        hashTable.insere(17081801, 410);
        hashTable.insere(17031121, 73);

        hashTable.getUpdateTree().emOrdem(hashTable, noAvl);
        hashTable.removeUpdateTree();

        System.out.println("Fluxo Minimo: " + noAvl.minFluxo());
        System.out.println("Fluxo Maximo: " + noAvl.maxFluxo());

        //min(fluxo) + 0.8Δ, onde Δ = max(fluxo) − min(fluxo).
        limiar = noAvl.minFluxo() + 0.8 * (noAvl.maxFluxo() - noAvl.minFluxo());
        System.out.println("Limiar: " + limiar);
        System.out.println("Saída: {setor,dia,fluxo}");

        noAvl.emOrdem(limiar);
    }
}
```