# Os megaprojetos

Douglas Paz[1]
Faculdade de Informática — PUCRS

6 de junho de 2018

Resumo

*Neste artigo será tratada uma solução para armazenamento e cálculo de custo de caminhamento utilizando grafos dirigidos e arestas valoradas, também será apresentada a detecção de ciclos*

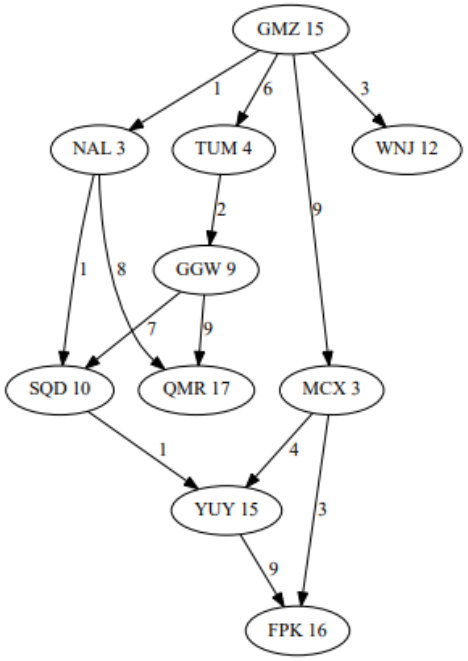
.

## Introdução

*Para este projeto foi apresentado o seguinte problema:*

> *"Sua empresa está desenvolvendo um complexo software de gerência de projetos que consegue calcular o custo de projetos compostos por várias atividades, com custos diferentes e em quantidades diferentes. O algoritmo para esta tarefa é sua responsabilidade e você deve ser capaz de analisar um projeto e determinar seu custo final. Um projeto é composto por atividades com custos próprios e uma atividade pode usar outras atividades: a atividade A pode custar $5 e ainda usar 3 vezes a atividade B (que custa $4), com um custo final para A de $17"*

Abaixo um exemplo gráfico de como seria organizada esta estrutura




Neste projeto cada atividade será denominada como "Nodo", e as quantidades de vezes que uma atividade tem dependente de outra será denominado como o peso de uma "Aresta".

Este artigo está separado através dos tópicos.

*Estrutura*: Onde contém como cada dado foi armazenado para ser computado posteriormente.

*Leitura*: Contendo de qual forma os dados foram transferidos dos arquivos de origem até as estruturas utilizadas no projeto.

*Cálculo de custo*: Exemplificando de qual forma se foi capaz de computar estes dados e atingir um cálculo preciso de resultados.

## Estrutura

Como primeira abordagem do problema os dados coletados dos arquivos de teste foram dispostos da seguinte forma:

Os primeiros dados a serem lidos correspondentes aos “Nodos” do grafo eram armazenados no seguinte objeto seguido de seus atributos:

```
public class Nodo {
    // estaticas
    private String nome;
    private int peso;
    private LinkedHashMap<String, Aresta> filhos = new LinkedHashMap<>();

    // dinamicas
    private char status;
    private BigDecimal pesoAcumulado;
```

Os valores de *nome* e *peso* foram serão coletados a partir da criação do objeto, o valor de *pesoAcumulado* será inicializado por padrão com o valor de peso do nodo e será explicado a seguir.

O dicionário *filhos* mostrado acima contém uma lista de um outro objeto mostrado abaixo:

```
public class Aresta {
    private int pesoAresta;
    private Nodo nodo;

    private BigDecimal pesoTotal;
```

O objeto “Aresta” como demonstrado acima contém nele o seu valor de peso (*pesoAresta)* recebido através da leitura dos dados e um objeto “Nodo”, a aresta também contém o valor denominado de *pesoTotal* calculado a partir do *pesoAcumulado* do nodo ali em questão multiplicado pelo *pesoAresta*.

## Leitura

Para leitura e instanciação foi realizada no seguinte método presente na classe “Maestro”:

```
LinkedHashMap<String, Nodo> leitura(String caminhoArquivo){
    LinkedHashMap<String, Nodo> grafo = new LinkedHashMap<>();
    try (BufferedReader reader =
        Files.newBufferedReader(Paths.get(caminhoArquivo))) {

        String[] conteudo;
        int index = Integer.parseInt(reader.readLine());

        // Leitura dos nodos
        for(int i = 0; i< index; i++) {
            conteudo = reader.readLine().split(" ");
            grafo.put(
                conteudo[0],
                    new Nodo(
                        conteudo[0],
                        Integer.parseInt(conteudo[1])));
        }

        index = Integer.parseInt(reader.readLine());

        // Leitura das arestas
        for(int i = 0; i< index; i++) {
            conteudo = reader.readLine().split(" ");
            if(grafo.containsKey(conteudo[0])
            && grafo.containsKey(conteudo[1])){

                grafo.get(conteudo[0])
                    .addFilho(
                        new Aresta(
                            Integer.parseInt(conteudo[2]),
                            grafos.get(conteudo[1])));
            }
        }
        return grafo;
    }
    catch (IOException x) {
        System.err.println("Arquivo para leitura de dados não
encontrado");
    }
    return null;
    }
```

```
}
```

O método exemplificado acima está dividido em duas partes principais, comentadas no código, a primeira lê no arquivo o primeiro segmento de dados e adiciona em um dicionário um novo grafo correspondente a aquele dado em questão, a segunda parte lê o no arquivo o segundo segmento de dados, busca o nodo correspondente a origem da aresta (o pai do nodo em questão correspondente a coluna ‘0’ da linha lida) e o correspondente ao destino e adiciona na origem um endereço ao destino como um novo *filho* e o seu respectivo peso de aresta, por fim, é retornado o dicionário de todos os nodos com seus respectivos filhos já adicionados.

# O Cálculo de custos

Na classe “Maestro” foi adicionado um método chamado *calculadoraGrafo,* este recebe o dicionário vindo da *leitura* dos dados, que é iterado, e a cada ciclo de iteração é chamado o método *getPesoAcumulado* de cada nodo do dicionário, que de fato efetua o cálculo de custo de cada nodo, por fim, será considerado raiz o nodo que conter o maior valor acumulado e este valor é retornado.

```
BigDecimal calculadoraGrafo(LinkedHashMap<String, Nodo> dados){
    BigDecimal resultado = new BigDecimal(BigInteger.ZERO);
    for(Nodo g: dados.values()) {
        BigDecimal pesoNodo = g.getPesoAcumulado();
        System.out.println(g.toString() + " -> " + pesoNodo);
        if(pesoNodo.compareTo(resultado) > 0)
            resultado = pesoNodo;
    }
    return resultado;
}
```

## Cálculo de Peso Acumulado

O método a seguir altera atributos do objeto “Nodo” apresentados anteriormente para o cálculo de seus custos.

```
public BigDecimal getPesoAcumulado(){
    if(status == 'O'){
        try {
            throw new CycleExpection(nome);
        } catch (CycleExpection cycleExpection) {
            System.err.println(cycleExpection);
            System.exit(1);
        }
    }
    if(status == 'N') {
        setStatusOcupado();
        if (temFilhos()) {
            for (Aresta a : filhos.values()) {
                pesoAcumulado = pesoAcumulado.add(a.getPesoTotal());
            }
        }
        setStatusVisitado();
    }
    return pesoAcumulado; // se for folha retorna o valor dele
}
```

Cada nodo contém um atributo de *status* que pode ser representado por apenas 3 valores: ‘N’ (novo), ‘O’ (ocupado), ‘V’ (visitado).

Assim que solicitado o peso acumulado de um nodo é verificado seu status, caso o mesmo esteja como ‘N’ (novo), ele será marcado como ‘O’ (ocupado) como forma de prevenção a ciclos por indicar que o nodo já está sendo utilizado atualmente, e caso seja solicitado um nodo ocupado a aplicação será interrompida por presença de ciclos, após isto será verificado sucessivamente o peso total de seus filhos (Arestas) e cada um deles somados ao seu peso acumulado.

```
public BigDecimal getPesoTotal() {
    return this.pesoTotal =
        nodo.getPesoAcumulado().multiply(new BigDecimal(pesoAresta));
}
```

Note que há um processo recursivo ao calcular o peso total devido que para calcula-lo é necessário o peso acumulado do nodo daquela aresta, que pode conter mais aresta dentro de si (filhos), este processo irá se repetir até que o nodo tenha somente filhos com status ‘V’ (visitado), por fim o próprio nodo será marcado como ‘V’ (visitado), pois seu peso acumulado agora já terá sido calculado.

## Conclusões

O projeto apesar de oneroso foi de suma importância para o entendimento de como estruturas complexas como grafos podem ser uteis para solucionar problemas algorítmicos que possam ser apresentados em alguma circunstância.
Para acompanhamentos futuros e atualizações sobre o projeto este relatório e os códigos fontes serão disponibilizados através do link: https://github.com/douglas-paz/Grafos