Algoritmos e Estruturas de Dados I
Trabalho 3 - As Esculturas

Daniele Ramos de Souza

28 de outubro de 2015

Resumo

Este trabalho é a terceira das avaliações da disciplina de Algoritmos e Estruturas de Dados I do curso de Engenharia de Software da Faculdade de Informática (FACIN) da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) e tem como objetivo desenvolver um algoritmo capaz de calcular o equilíbrio de uma série de esculturas.

1 Introdução

O desafio deste trabalho é desenvolver um algoritmo capaz de calcular o equilíbrio de uma série de esculturas, pertencentes a um ousado projeto de arquitetura, as quais devem ficar em pé apenas por causa da gravidade. Estas esculturas possuem braços com encaixes para outros braços ou com pesos que permitem o balanceamento da obra, como podemos visualizar no exemplo de escultura da Figura 1.

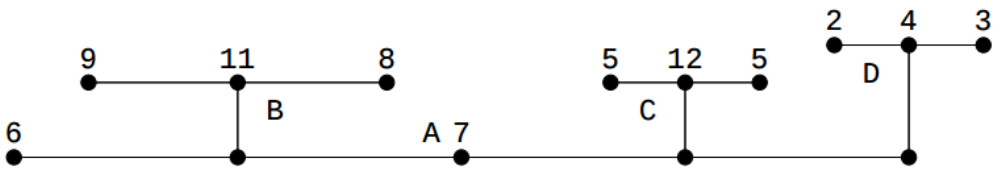

Figura 1: Exemplo de escultura

Para testar se o nosso algoritmo funciona, recebemos arquivos de entrada que fornecem as seguintes informações para compor uma escultura: o número de braços que a escultura possui e as informações necessárias sobre cada braço (nome, número de encaixes e valor dos encaixes - peso ou nome do braço que segura). O arquivo de entrada do exemplo de escultura da Figura 1 seria assim, por exemplo:

4						
B	3	9	11	8		
A	5	6	B	7	C	D
D	3	2	4	3		
C	3	5	15	5		

Tabela 1: Exemplo de arquivo de entrada

2 Estrutura de dados

Para armazenar os dados dos arquivos de entrada, vamos utilizar uma estrutura de árvore. A árvore será composta por um elemento Nodo, que será sua raiz.

Cada Nodo da nossa árvore armazena as seguintes informações em forma de atributos: um nome (no caso de ser um novo braço da estrutura), um valor (no caso dos encaixes que possuem peso), a sua posição e sua distância em relação ao encaixe central do braço. Além disso, cada Nodo possui uma lista encadeada de Nodos para armazenar seus filhos.

A figura 2 mostra como o exemplo de escultura da figura 1 é armazenado em uma árvore.

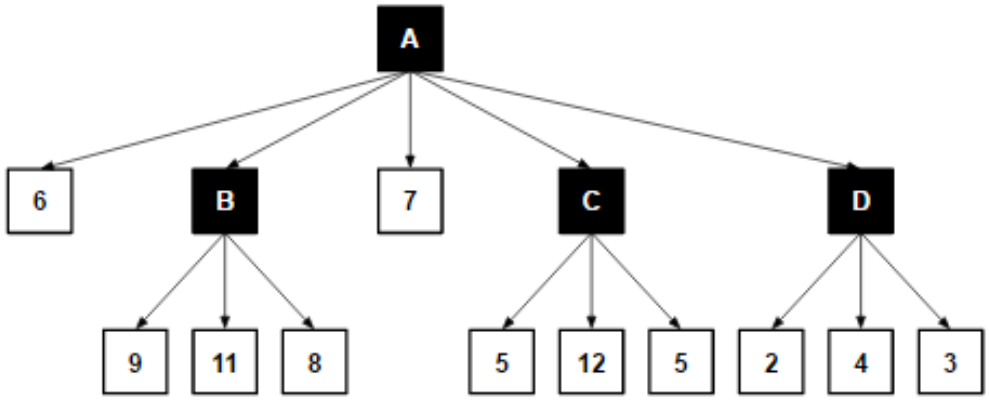

Figura 2: Árvore do exemplo de escultura

3 Lendo os arquivos

Pegando como o exemplo o arquivo de teste da tabela 1, sua leitura é feita da seguinte maneira:

A primeira linha representa o número de braços que a escultura possui. Ela é lida e o dado é armazenado em uma variável (neste caso, vale 4).

A partir da segunda linha, temos o mesmo padrão descrevendo as informações sobre os braços da estrutura (seu nome, o número de encaixes que ela possui e quais são eles). Desta forma, enquanto houverem linhas a serem lidas, pegamos uma e quebramos as informações contidas nela separadas por espaços em branco. Com estas informações, somos capazes de criar nodos e adicionar filhos a um nodo.

Como alguns braços da estrutura possuem encaixes para outros braços, muitas vezes as informações sobre estes braços ainda não foram lidas, criando uma dependência entre eles e, conforme a ordem em que estão inseridas nos arquivos de teste, podem ou não limitar a inserção deles na estrutura de árvore. Para resolver este problema, utilizamos um HashMap que guarda os braços que já foram lidos anteriormente, que possui como chave o nome de um braço e como valor uma referência para o respectivo nodo.

A vantagem se usar HashMap é que podemos ir construindo a árvore ao mesmo tempo em que vamos lendo o arquivo. Uma outra opção seria ler todo o arquivo primeiro e armazenar todas as informações e depois, a partir destas informações, ir construindo a árvore. Porém, nesta segunda opção, além de lermos uma vez o arquivo, precisaríamos realizar muitas pesquisas nas informações armazenadas, enquanto que no HashMap seu custo é muito inferior, pois, sabendo o nome do braço, podemos acessar diretamente suas informações. Além disso, reforça-se o fato de que com o HashMap podemos construir a árvore ao mesmo tempo em que vamos lendo o arquivo.

Se estamos lendo uma linha com informações sobre um braço pela primeira vez, então simplesmente criamos um novo Nodo, que será a raiz da árvore. É o caso da segunda linha do arquivo de teste que estamos pegando como exemplo. Criamos um Nodo com nome B e populamos sua lista de filhos com as informações seguintes.

Se já lemos alguma linha anteriormente, verificamos se um braço da estrutura possui um encaixe para outro braço que já existe como um Nodo da árvore no nosso HashMap. Neste caso, chamamos sua referência e fazemos o devido relacionamento (além de alterar a raíz da árvore para ser o novo nodo). Caso contrário, criamos um novo Nodo e adicionamos ele à lista, com as informações que possuímos sobre ele até o momento (nome, posição e distância do centro). Quando a linha deste for lida, encontramos ele no HashMap e atualizamos suas informações.

Este algoritmo de leitura pode ser visualizado, em formato de pseudocódigo, no Apêndice A deste documento.

4 Calculando o peso de um braço

No cálculo do peso de um braço, a distância dos encaixes em relação ao centro do braço não importa. Portanto, a única coisa que devemos fazer é percorrer a lista de filhos de um nodo e ir somando seus valores. Se um nodo filho for terminal (um valor de peso), somamos seu valor a um contador. Caso ele não seja terminal (uma letra), chamamos o mesmo método recursivamente para obter seu valor de peso.

Na Figura 1, por exemplo, o cálculo de peso do braço B é efetuada da seguinte maneira: $peso = 9 + 11 + 8 = 28$. Para descobrirmos o peso do braço A, teríamos que chamar recursivamente o método para calcular o peso dos braços B, C e D, totalizando em 65.

O algoritmo que calcula o peso de um nodo pode ser visualizado no pseudocódigo abaixo:

```
peso_nodo():

    total = 0

    para todos os filhos
        se filho é terminal
            valor = filho.valor
        se não
            valor = filho.peso_nodo()

        total += valor

    retorna total
```

5 Calculando o equilíbrio de um braço

Para calcular o equilíbrio de um braço da estrutura, devemos levar em consideração a distância dos seus encaixes em relação ao encaixe central. Quanto mais próximos das laterais de um braço, mais os pesos dos encaixes impactam no balanceamento do mesmo.

Na Figura 1, o encaixe de peso 6 e o encaixe que suporta o braço D do braço A têm peso 2, pois encontram-se a 2 encaixes de distância do encaixe central. O encaixe que suporta o braço B e o encaixe que suporta o braço C, que estão mais próximos do encaixe central do braço A, têm peso 1. O peso do encaixe central, por sua vez, não deve ser considerado para o cálculo do balanceamento de um braço, pois não cria uma tendência para nenhum dos lados.

Então, para saber se um braço está em equilíbio devemos verificar se a soma dos pesos dos encaixes à esquerda do centro do braço é igual à soma dos pesos dos encaixes à direita, ambos multiplicados aos seus respectivos pesos.

Na Figura 1, por exemplo, o cálculo de balanceamento do braço A é dada da seguinte maneira (chegando à conclusão de que o braço A está balanceado):

$$(6 * 2) + (B * 1) = (C * 1) + (D * 2)$$
$$12 + ((9 + 11 + 8) * 1) = ((5 + 12 + 5) * 1) + ((2 + 4 + 3) * 2)$$
$$12 + 28 = 22 + 18$$
$$30 = 30$$

O algoritmo que calcula o equilíbrio de um braço pode ser visualizado no pseudocódigo a seguir:

```
balanceado_nodo():
    esquerda = 0
    direita = 0

    para todos os filhos
        se filho é terminal
            valor = filho.valor
        se nao
            valor = filho.peso_nodo()

        se o filho estiver à esquerda
            esquerda += valor * filho.multiplicador
        se o filho estiver à direita
            direita += valor * filho.multiplicador


    retorna esquerda == direita
```

5.1 Resultados

A tabela 2 apresenta os resultados dos testes.

Caso de teste	Número de braços	Em equilíbrio	Fora de equilíbrio
1	155	36	119
2	142	37	105
3	167	32	135
4	176	29	147
5	159	33	126
6	118	28	90
7	195	49	146
8	204	52	152
9	254	59	195
10	98	15	83

Tabela 2: Resultados dos casos de teste

A tabela 3 apresenta os tempos de execução de cada caso de teste.

Caso de teste	**Tempo de execução**
1	0.0137 segundos
2	0.0051 segundos
3	0.0039 segundos
4	0.0032 segundos
5	0.0021 segundos
6	0.0015 segundos
7	0.0021 segundos
8	0.0019 segundos
9	0.0021 segundos
10	9.9709E-4 segundos

Tabela 3: Resultados dos casos de teste

6 Considerações

A escolha da estrutura de dados para solucionar um problema é de extrema importância, pois pode inviabilizar por si só a solução de um problema. Cada estrutura tem seus prós e contras, como tempo de busca, tempo de inserção, entre outros.

Nesta implementação, a escolha de se utilizar uma árvore facilitou o desenvolvimento dos algoritmos, visto que a escultura já tinha naturalmente este "formato"e que o cálculo de peso e balanceamento de um braço poderia vir a depender do cálculo de outro braço, caracterizando uma dependência. Acredito que a escolha de árvore como estrutura de dados foi feliz, pois foi fácil de tratar tais dependências.

Além disso, nosso algoritmo de leitura foi capaz de "burlar"algumas vezes o tempo de inserção em uma árvore com o auxílio de um HashMap. Guardando a referência de nodos de dependência, evitamos de realizar diversas buscas por nossa estrutura à procura da mesma.

Contudo, a estrutura de dados por si só não garante que um algoritmo execute de maneira eficiente (tempo e memória). Pensando nisso, procuramos aproveitar ao máximo da estrutura de dados escolhida ao desenvolvermos os algoritmos.

Uma melhoria a ser feita no programa atual poderia ser, durante a execução do cálculo de balanceamento geral, guardar os resultados obtidos até agora em alguma estrutura (um HashMap, por exemplo). Atualmente, pegando como exemplo a Figura 1, iremos calcular duas vezes o peso dos braços B, C e D, uma vez para calcular seu balanceamento e outra para calcular o balanceamento de seu pai, o braço A.

Apêndice A - Leitura de Arquivo

```
leitura_arquivo():

    lidos = hash(nodo)

    primeira_vez = true

    numero_braços = ler_linha()

    enquanto houverem linhas a serem lidas

        linha = quebra_linha(ler_linha(), " ")

        nome_braco = linha[0]

        numero_encaixes = linha[1]

        se lidos.contém(nome_braço)
            braço = lidos.pega(nome_braço)
        se não
            braço = Nodo(nome_braço)


        se primeira_vez
            arvore.raiz = braço
            primeira_vez = false

        centro = numero_encaixes/2

        distancia_do_centro = -centro

        para i de 2 até comprimento(linha)

            se distancia_do_centro < 0
                posicao = esquerda
            se distancia_do_centro == 0
                posicao = meio
            se distancia_do_centro > 0
                posicao = direita

            multiplicador_distancia = modulo(distancia_do_centro)

            se o valor de linha[i] for inteiro
                valor = linha[i]
                aux = nodo(valor, posicao, multiplicador_distancia)

            se nao
                se lidos.contém(linha[i])
                    aux = lidos.pega(linha[i])

                    aux.posicao = posicao
                    aux.multiplicador = multiplicador_distancia
```

```
                se arvore.raiz == aux
                    arvore.raiz = braço

            se não
                aux = nodo(linha[i])

                aux.posicao = posicao
                aux.multiplicador = multiplicador_distancia


        braço.adiciona_filho(aux)

        distancia_do_centro += 1

    lidos.coloca(braço)

retorna árvore
```