Pontifícia Universidade Católica do Rio Grande do Sul
Escola Politécnica

Algoritmos e Estruturas de Dados II - Turma 128
Professor Diego Vrague

Trabalho 1

Adriano Ramos, Artur Marcon e João Cervo

adriano.ramos@edu.pucrs.br
a.marcon@edu.pucrs.br
j.dourado@edu.pucrs.br

Abril de 2021

1. Introdução:

O objetivo deste relatório é apresentar a solução proposta para o primeiro trabalho da disciplina de Algoritmos e Estruturas de Dados II, ministrada pelo Professor Diego Vrague Noble.O trabalho consiste em balancear o esforço de trabalho de um conjunto de computadores para não gerar sobrecarga de tarefas para um único computador e interferir nos seus desempenhos.

Com o que foi estudado até o presente momento, nossa solução foi baseada em desenvolver um algoritmos de Min Heap que avalia o valor de trabalho em cada nodo e compara se está balanceado, ou não, de acordo com seus nodos filhos. A implementação do Min Heap e seus métodos pode ser visualizada na figura 1.

```java
public class MinHeap {
    public ArrayList<Computador> elems;
    private int size;

    public MinHeap() {…

    /**
     * Insere um computador no heap.
     * @param element o computador a ser inserido.
     */
    void insert(Computador element) {…

    /**
     * Verifica quantos elementos são balanceados
     * @return o numero de elementos balanceados
     */
    int howManyAreBalanced() {…

    /**
     * Pega o ultimo indice do computador ainda com nome
     * Ex: 'X30', sendo os filhos dele computadores sem nome
     * @return o indice do ultimo computador com nome
     */
    int getLastComputerWithName() {…

    @Override
    public String toString() {…
}
```

Figura 1: Implementação do Min Heap

Fonte: Autoria Própria

2. Solução

Para resolver este problema, implementamos uma classe 'Computador', que contém todos os atributos de um objeto computador, como: nome e trabalho, seguido de alguns métodos de acesso, também foi implementado um leitor de arquivos txt para lermos os arquivos de texto contendo os casos de teste, e por fim, o algoritmo central que foi desenvolvido para solucionar o problema proposto foi um Min Heap.

Seguindo a ideia de dividir para conquistar, foi desenvolvido um método que verifica quantos nodos estão balanceados. Nesse, primeiramente pegamos o último computador a ter um nome, isto é, que tem o seu trabalho compartilhado com outros dois. Então

percorremos a partir deste até a metade (índice do último computador com nome / 2 - 1), pois como estamos usando uma estrutura Min Heap, conseguimos pegar o nodo dos filhos usando índice * 2, seguido de + 1 para filho da esquerda e + 2 para filho da direita, vendo quais computadores tem os nodos balanceados de baixo para cima. Todos os computadores lidos aqui são adicionados em uma estrutura de HashMap, para melhor identificação de quais computadores já foram calculados. Assim conseguimos calcular quais nodos no topo estão balanceados pegando o trabalho desses nodos filhos já calculados, evitando percorrer desnecessariamente mais computadores do que precisamos. Após isto, percorremos entre os computadores sem nome, calculando o trabalho destes e vendo se são balanceados. Na figura 1 está um print da implementação desta parte. Posteriormente, percorremos pelo dicionário de computadores calculados, contando quantos eram balanceados, isto é, o primeiro item no array de values tinha valor 2. Este processo pode ser visto na figura 3.

```
HashMap<String, int[]> computersWithWorkCalculated = new HashMap<>();

int indexOflastComputerWithName = getLastComputerWithName();
int half = indexOflastComputerWithName / 2 - 1;

for (int i = indexOflastComputerWithName; i > half; i--) {
    Computador thisComputer = elems.get(i);

    Computador leftChild = elems.get(i * 2 + 1);
    Computador rightChild = elems.get(i * 2 + 2);

    int work = leftChild.getTrabalho() + rightChild.getTrabalho();
    int isBalanced = leftChild.getTrabalho() == rightChild.getTrabalho() ? 1 : -1;

    int[] values = { isBalanced, work };
    computersWithWorkCalculated.put(thisComputer.getNome(), values);
}

for (int i = half; i >= 0; i--) {
    Computador thisComputer = elems.get(i);

    String leftChildName = "X" + (i * 2 + 1);
    String rightChildName = "X" + (i * 2 + 2);

    int[] leftChild = computersWithWorkCalculated.get(leftChildName);
    int[] rightChild = computersWithWorkCalculated.get(rightChildName);

    int isBalanced = leftChild[1] == rightChild[1] ? 1 : -1;
    int work = leftChild[1] + rightChild[1];

    int[] values = { isBalanced, work };
    computersWithWorkCalculated.put(thisComputer.getNome(), values);
}
```

Figura 2: Implementação do algoritmo de balanceamento dos nodos.

Fonte: Autoria Própria

```
int balancedElements = 0;
Iterator it = computersWithWorkCalculated.entrySet().iterator();
while (it.hasNext()) {
    Map.Entry pair = (Map.Entry)it.next();
    if (computersWithWorkCalculated.get(pair.getKey())[0] == 1) {
        balancedElements++;
    }
    it.remove();
}
return balancedElements;
```

Figura 3: Contagem dos nodos balanceados.

Fonte: Autoria Própria

3. Casos de Teste

Os casos de testes utilizados foram os enviados pelo professor, Diego Noble. Neste, temos nove arquivos txt, cujo nome vai de teste5 a teste13. Estes nomes serão usados para se referenciar a cada teste na tabela 1. Como descrito anteriormente, todos os casos de testes foram implementados utilizando um leitor de txt, implementado por nós. A abordagem dividir para conquistar descrita na seção anterior nos possibilitou uma execução muito rápida de todos os casos, apesar dos arquivos serem longos, todos foram executados instantaneamente. Segue abaixo a tabela com os resultados obtidos em cada teste.

Casos de Teste	Quantidade de nodos balanceados
Teste 5	6
Teste 6	9
Teste 7	18
Teste 8	33
Teste 9	53
Teste 10	78
Teste 11	118
Teste 12	134
Teste 13	169

Tabela 1: Resultados dos casos de teste.

Fonte: Autoria Própria.

4. Conclusão

O desenvolvimento do presente trabalho nos possibilitou maior exploração e fixação dos conteúdos abordados em aula. Inicialmente, o problema nos fez debater diversas formas de solução e abordagens que poderíamos utilizar para implementação do algoritmo, isso nos levou a testar e estudar algumas das estruturas de dados já estudadas em aula para então definir uma que se encaixasse no contexto deste desafio. Também tivemos algumas dificuldades no projeto, principalmente em como iríamos realizar a contagem dos nodos balanceados de maneira eficiente.

Levou um tempo até que chegássemos ao algoritmo descrito na seção 2, e foram testadas diversas estruturas diferentes. Primeiramente, utilizamos uma árvore binária, porém o fato de ter um acesso mais lento aos elementos não serviu a maneira que estávamos tentando propor a solução. Em seguida, se deu a utilização de um array, mas tivemos problemas em organizar os elementos de uma forma coesa e que fosse eficiente. Então chegamos à solução de construir um Min Heap para armazenar os dados. Por fim, o desenvolvimento deste trabalho nos possibilitou bastante aprendizado e foi uma experiência positiva para todos do grupo.