Trabalho I - Salvando as árvores

Bianca Camargo Machado*
Escola Politécnica - PUCRS

3 de outubro de 2018

Resumo

Este artigo descreve uma solução para o primeiro problema proposto na disciplina Algoritmos e Estruturas de Dados II no semestre 2018/2, trata-se da determinação da altura das ávores binárias que guardam segredos do Greenpeace. Foram disponibilizados 10 casos de teste para resolução do problema: cada um contém duas listagens de uma mesma árvore binária, sendo uma delas em caminhamento pré-fixado e outra em caminhamento central. Após descrever a solução do problema, são apresentados os resultados obtidos para os casos de teste, juntamente com seus tempos de execução.

Introdução

No contexto da disciplina de Algoritmos e Estruturas de Dados II, o enunciado do primeiro trabalho proposto pode ser resumido da seguinte forma: depois de ter em mãos o documentos secretos do Greenpeace e analisada a forma como estes documentos são codificados, constata-se que os documentos são armazenados em forma de árvores binárias e que para cada árvore existem duas formas de listagem: uma com caminhamento pré-fixado e outra com caminhamento central. Tendo as duas listagens referentes a uma mesma árvore dispostas em um arquivo, assume-se o objetivo de determinar a altura da árvore descrita nas listagens.

Após observado o problema e seus casos de teste, as primeiras informações que podemos listar sobre ele são as seguintes:

- Todo arquivo contém informações sobre uma única árvore;
- As listagens estão dispostas em duas linhas, sendo que a primeira é a com caminhamento pré-fixado e a segunda com caminhamento central;
- Não há tamanho máximo e mínimo para as listagens;
- Não é necessário diferenciar letras maiúsculas e minúsculas;
- O conteúdo do arquivo se restringe a conjunto de palavras formados por letras e números;
- Cada palavra não se repete na mesma listagem do caso de teste, ou seja, é encontrada epenas uma vez na listagem pré-fixada e epenas outra vez na listagem infixada;
- Os valores `null` não fazem parte das listagens.

A fim de ilustrar o problema de maneira prática, utilizaremos uma versão simplificada do padrão de listagem encontrada nos documentos secretos do Greenpeace: consideraremos que as informações contidas na árvore são letras do alfabeto de A a K e segue-se todos os padrões observados e listados acima. Assim como os valores `null` não aparecem nas listagem, também não serão mostrados nas imagens referentes às árvores, já que não influenciam na altura da ávores.

Abaixo encontram-se exemplos de listagem e a ilustração da árvore que corresponde a elas:

A B D H I E C F G J K

H D I B E A F C J G K

*bianca.camargo@acad.pucrs.br

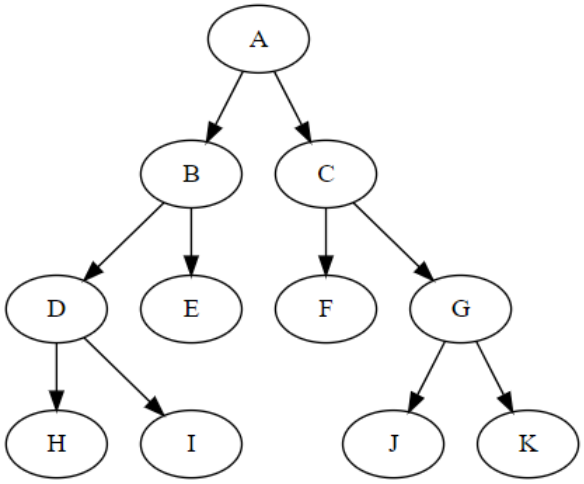

Figura 1: Árvore correspondente às listagens exibidas acima.

Segundo [1] , o caminhamento de uma árvore T é uma forma sistemática de acessar ou "visitar" todos os nodos de T. Juntamente com este conceito cabe destacar que caminhamento pré-fixado de uma árvore ocorre da seguinte forma: visita a raiz, percorre sub-árvore esquerda e percorre sub-árvore direita, nesta ordem. Já o caminhamento infixado ou central ocorre da seguinte forma: percorre sub-árvore esquerda, visita a raiz e percorre sub-árvore direita, nesta ordem.

O objetivo principal do trabalho é calcular a altura da árvore presente nas listagens, sabemos que o algoritmo que realiza este cálculo é relativamente simples e geralmente abordado em sala de aula durante o estudo da estrutura de árvores. Sabemos também que a altura da árvore é a distância da sua raiz até seu nodo mais afastado, pode-se dizer também que é o número de arestas que necessita-se percorrer para chegar ao nodo mais afastado da raiz.

No exemplo de árvore mostrado acima a altura é 3, já que os nodos mais afastados de A (raiz) são H, I, J e K, e para chegar em qualquer um deles, a partir de A, é necessário percorrer três arestas. Se para chegar a qualquer um deles fosse necessário percorrer um número de arestas maior do que este, o mesmo seria a altura da árvore, já que considera-se a maior distância necessária.

Sabendo de todas estas informações podemos observar que um empecilho para determinarmos a altura da árvore de cada listagem é a falta conhecimento sobre a estrutura completa da mesma, pois não temos diretamente, por exemplo, o número de arestas e os nodos aos quais elas se ligam. Portanto, considerando o exemplo dado, o objetivo resume-se a determinar a altura da árvore apenas com listagens como **A B D H I E C F G J K** e **H D I B E A F C J G K** e as informações que as cercam.

A seguir será explicado como foi construído o algoritmo que soluciona o problema, a lógica abordada na resolução e, por fim, a desmonstração dos resultados obtidos para os casos de testes diponibilizados junto ao problema.

Análise do problema

De posse das informações contidas na introdução, ou seja, conhecendo os meios e informações disponíveis para solucionarmos o problema, é possível notar que sendo as duas listagens (dispostas no mesmo arquivo) referências de uma única árvore, é possível chegarmos a conclusões relacionadas à estrutura da mesma. Mas de que forma? A resposta é: através dos padrões estabelecidos nos caminhamentos. Um exemplo disto é a listagem dada na Introdução, da qual podemos extrair que o primeiro elemento contido na listagem pré-fixada será sempre a raiz da árvore, já que respeita-se a sistemática do caminhamento da mesma. Portanto, podemos concluir que a raiz da árvore é o A, sem a necessidade de visualizar a graficamente a estrutura da mesma.

Seguindo este raciocínio, podemos observar que a raiz da árvore, na listagem infixada, estará no centro da mesma (exceto quando a quantidade de elementos da esquerda e direita forem diferentes) e ainda: tendo o nodo raiz da árvore a partir da primeira listagem, podemos localizá-lo na segunda e determinar quais nodos pertecem à esquerda da árvore e quais nodos pertencem à direita, utilizando-se do fato de que o caminhamento infixado segue a ordem sub-árvore esquerda, raiz e sub-árvore direita.

Para ilustrar esta ideia observe a imagem a seguir:

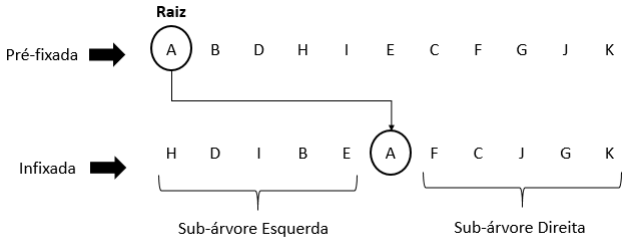

Figura 2: Demonstração da relação entre o primeiro elemento da listagem pré-fixada (raiz) e a listagem infixada.

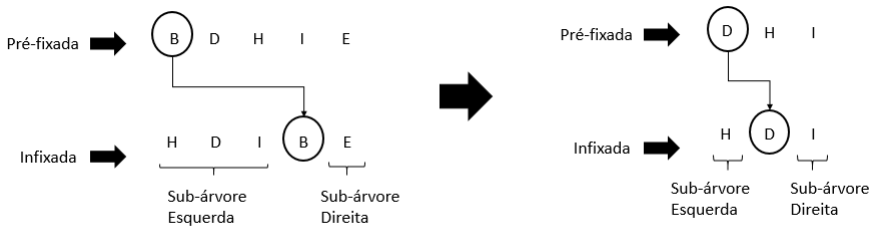

Figura 3: Fluxo de recursão resumido para acompanhar a lógica de definição da estrutura da árvore.

Podemos notar que este raciocínio pode nos levar à estrutura completa da árvore já que, recursivamente, repetindo estes mesmos passos, podemos chegar a estrutura completa da árvore, para que ao final possamos calcular a altura da mesma. Acompanhe o fluxo de repetições a seguir, que ilustra esta ideia:

A imagem acima demonstra um fluxo resumido de como se dá o processo de estruturação da árvore do lado esquerdo a partir da primeira divisão, seguindo os seguintes passos: determinação da raiz, localização da mesma na listagem infixada e a separação da sub-árvore em sub-árvore esquerda e direita. O processo é repetido até que a mesma esteja com sua estrutura completa (até chegarmos à estrutura de árvore exibida na Figura 1 - Página 1). Vale destacar que, para simplificar a demonstração, o mesmo processo ocorre nas sub-árvores resultantes de **F C J G K** (Figura 2), mas o mesmo não é exibido na imagem acima por tratar-se do mesmo processo desmontrado a partir de **H D I B E**.

Podemos observar que H e I são filhos de D através desta divisão.

Solução

O algoritmo para o problema em questão foi implementado na linguagem Java e segue a lógica descrita anteriormente. Os casos de testes estavam disponíveis em um arquivo de texto, portanto o algoritmo que calcula a altura da árvore foi implementado após a leitura do arquivo e armazenamento das listagens em duas listas separadas, uma contendo a listagem com caminhamento pré-fixado e outra contendo a listagem com caminhamento infixado. Para a resolução do problema optou-se por utlizar principalmente dois métodos: um método para montar a estrutura da árvore (nodo, esquerda e direita) e outro para calcular a altura da mesma.

O método que monta a estrutura da árvore retorna a raiz da árvore, para que seja utilizada no método que calcula sua altura. Este algoritmo percorre as posições na listagem pré-fixada e busca cada elemento na listagem infixada. Tendo a posição do elemento, a qual chamaremos de **marca** (Exemplo: na Figura 2 a marca de A na listagem infixidada é 5, posição na qual elemento se encontra). A partir desta posição podemos dividir a árvore em esquerda e direita, como descrito na análise do problema. Para controlar quando a determinação da estrutura de sub-árvores da esquerda e da direita foram finalizadas, o método recebe como parâmetro duas informações de onde ele deve inciar a comparação e onde deve parar. Isto para que seja possível garantir que a procura de nodos do lado esquerdo está sendo feita somente nele e o mesmo para o lado direito. E sem ultrapassar o limite da sub-árvore. A seguir estão alguns pontos importantes do algoritmo:

- Quando se está determinado as sub-árvores da esquerda: a posição incial deve ser 0 (na primeira

repetição) e a final deve ser a **marca** -1, já que o nodo pai não deve ser divido novamente, mas sim servir de marca a a divisão da sub-árvore;

- Quando se está determinando as sub-árvores da direita: a posição inicial deve ser **marca** +1, já que o nodo pai também não deve ser considerado na nova divisão, e a final de ser incrementado desde a posição da marca até no máximo o tamanho da lista;
- O valor da **marca** é atualizado a cada repetição, já que o índice que percorre a listagem pré-fixada é incrementado de 1 a 1 até chegar ao final da listagem;
- Quando o índice inicial for maior que o tamanho do índice final (passados por parâmetro) significa que chegamos a um nodo `null`,cheste que não exerce influência sobre a altura da árvore;
- Quando o índice inicial for igual a índice final significa que chegamos a um nodo folha (um dos exemplo é o nodo I - Figura 1).

O método apresentado a seguir tem como objetivo definifir a estrutura da árvore a partir das de listagens fornecidas, utilizando-se de todos os conceitos explicados acima. Considera-se também a classe Arvore, com a seguinte estrutura: três atributos, esquerda, direita e elemento, sendo dois deles do tipo Arvore e o último do tipo String, além do construtor da classe.

```
//Na primeira execução de = 0 e para = (tamanho da listagem)-1
estruturaArvore(int de, int para) {

    //Critério para definição de nodo folha nulo
    se (de > para)
        retorna null;
    String nodoPreFixada = listagemPre.get(indice++);
    Arvore subarvore = nova Arvore(nodoPreFixada);
    //Chegamos a uma extremidade (nodo folha)
    se (de == para)
        retorna subarvore;
    //Marcamos a posição do elemento de indíce atual na listagem infixada
    int marca = listagemInfixada.indexOf(subarvore.elemento);
    //Buscamos o nodo da esquerda, passando os novos índices para localizá-lo
    subarvore.esquerda = estruturaArvore(de, marca-1);
    //buscamos o nodo da direita, passando os novos índices para localizá-lo
    subarvore.direita = estruturaArvore(marca+1, para);
    //Após gerada a estrutura da árvore: o nodo raiz é retornado
    //A partir do qual é realizado o cálculo da altura
    retorna subarvore;

}
```

A partir do retorno do nodo raiz pelo método listado acima, o método responsável por calcular a altura da árvore é o srguinte:

```
//cálculo da altura
altura(Arvore elemento) {

    se (elemento == null)
        retorna -1;
    senão {
        int alturaEsquerda = altura(elemento.esquerda);
        int alturaDireita = altura(elemento.direita);
        //retorno a maior altura, esquer ou direita
        se (alturaEsquerda < alturaDireita)
            retorna ++alturaEsquerda;
        senão retorna ++alturaDireita;
    }
```

```
}
```

Os métodos apresentados acima foram fundamentais na resolução do problema abordado neste artigo, E uma das formas de testar se o resultado era o esperado foi utilizando a árvore citada como exemplo durante a descrição do problema (Introdução e Análise do problema), a partir da qual obtivemos o resultado de altura 3. Além deste exemplo, também validou-se a seguinte situação: sendo cada uma as listagens, infixada e pré-fixada, compostas por um único elemento, o resultado esperado como altura da árvore seria 0. E o mesmo foi obtido.

Resultados

Os resultados obtidos com os casos de testes, disponibilizados junto ao problema, exigiram um tempo de execução do algoritmo que variou de 44 milisegundos a 36.834 milisegundos e esta diferença de tempo deve-se a complexidade da estruturadas linstagens contidas em cada caso.

Caso de teste	Resultado Obtido (altura da árvore)	Tempo de execução do algoritmo (milissegundos)
caso05	14	44
caso10	15	48
caso15	20	101
caso20	26	339
caso25	24	322
caso30	30	2.481
caso35	26	899
caso40	32	12.051
caso45	33	9.494
caso50	34	36.834

Tabela1: Resultados dos casos de testes aplicados ao algoritmo desenvolvido.

Conclusões

A solução desenvolvida e apresentada ao longo deste artigo obteve resultados satisfatórios em relação aos resultados esperados e o tempo necessário para que o algoritmo resolvesse os problemas disponíveis nos casos de teste. Tendo em vista que algumas decisões a nível lógico e estrutural do código podem afetar de forma significativa os resultados e efeciência do algoritmo, foi de extrema importância o entendimento do problema como um todo e detalhamento das informações que inerentes à sistemática dos caminhamentos pré e infixidado de árvores binárias. Possibilitando chegar-se a resultados satisfatórios e o obajetivo principal concluído.

Referências

[1] Goodrich, Michael T.: “Estuturas de Dados e algoritmos em Java”. 5 ed. Bookman, 2013, 713 p.