XTree

Fernando Elger
Engenharia de Software — PUCRS

15 de setembro de 2019

Resumo

Este artigo descreve alternativas de solução para o primeiro problema proposto na disciplina de Algoritmos e Estruturas de Dados II no semestre 3, que trata da análise de uma XTree, que por definição é uma maneira mais compacta de representar árvores genéricas. É apresentado a lógica por trás da XTree e o funcionamento do algoritmo que averigua a integridade da árvore, além de avaliar sua eficiência. Em seguida é exposto os resultados para oito casos analisados.

Introdução

Dentro do escopo da disciplina de Algoritmos e Estruturas de Dados II, o primeiro problema pode ser resumido assim: acreditamos que conseguimos criar uma maneira mais compacta para representar árvores genéricas, uma vez que está nos incomodando arquivos XML cada vez maiores e com desperdício de bytes. Nossa solução é a XTree, uma árvore que garante o uso de arquivos menores e uma vantagem extra do sigilo.

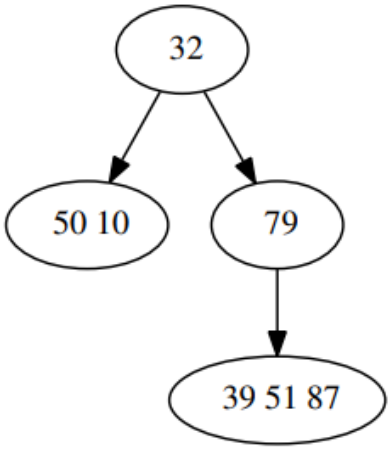
A XTree é baseada em três regras simples:

1. Um nodo não tem nome. Sua descrição começa com um cabeçalho composto por dois inteiros:
    a. Um inteiro $f \geq 0$ indicando quantos filhos ele tem;
    b. Um inteiro $n \geq 1$ indicando quantos valores inteiros o nodo contém, pois cada nodo pode conter mais de um valor.
2. Depois do cabeçalho segue a descrição dos f nodos filhos;
3. Depois seguem-se os n valores inteiros contidos no nodo.

Por exemplo, a descrição de uma XTree é dada por

2 1 0 2 50 10 1 1 0 3 39 51 87 79 32

Ela indica um nodo om 2 filhos e uma valor (32). Um desses dois filhos tem 0 filhos e dois valores (50 e 10) enquanto o outro filho tem um filho e um valor (79). O nodo que está mais fundo na árvore tem 0 filhos e três valores (39, 51 e 87).



Representação da XTree do exemplo

O problema a ser resolvido é elaborar um programa capaz de ler a descrição de uma XTree, que está em um arquivo de texto, e descobrir estas informações que, segundo o enunciado, é considerado um bom teste:

1. A quantidade de nodos da XTree;
2. A altura da XTree;
3. A soma de todos os valores contidos nos nodos.

No caso do exemplo acima, as informações encontradas devem ser sucessivamente:

1. 4 nodos;
2. Altura 2;
3. 348.

Para resolver o problema proposto, analisaremos uma primeira solução com imprecisões e em seguida uma possível solução funcional, bem como suas características e dificuldades encontradas. Em seguida os resultados obtidos dos casos serão apresentados, bem como as conclusões obtidas no decorrer do trabalho.

Primeira solução

Depois de considerar o problema, podemos supor que a descrição de uma XTree se comporta de maneira ambígua. Começamos a suspeitar da ideia de que uma suposta lista com os dados poderia ser consumida do início e do fim ao mesmo tempo. Essa ideia se sustenta se considerarmos que quando um nodo não é uma folha, seu elemento se encontra no final da lista.

Após isso, fica claro que devemos começar a pensar em uma maneira de ler e lidar com as descrições das XTrees, que se encontram em arquivos de texto onde cada elemento das descrições estão separados por espaços simples. Concluímos que devemos usar o Scanner para ler o arquivo e, em seguida ir adicionando cada elemento dentro de uma lista encadeada de inteiros para não ter que lidar com uma conversão posteriormente, já que teríamos que fazer a soma de alguns desses elementos. A lista encadeada foi escolhida pois, o algoritmo que iria analisar a XTree faria sempre a

remoção do primeiro e do último elemento da lista conforme ela iria sendo consumida, uma vez que os métodos de remoção da lista encadeada possuem um tempo de execução constante.

Não precisou ser testado nenhum caso para percebermos que a ideia de consumir uma lista do início e do fim ao mesmo tempo não iria se manter, pois poderia haver alterações na integridade dos dados dos nodos se a descrição da XTree fosse maior. Analisando firmemente as regras da XTree, podemos concluir que devemos consumir a lista de elementos do início e de forma contínua, pois quando um nodo não é uma folha, seu devido elemento será consumido naturalmente, assim que todas as informações de seus filhos tiverem sido consumidas.

Segunda solução

Uma vez que a primeira alternativa se provou insustentável em resolver o problema, já que devemos consumir os dados da lista do início e de forma contínua, optamos primeiramente por desenvolver uma abordagem mais simples. Considerando ainda que iremos continuar usando a lista encadeada para armazenar os elementos da descrição da XTree e que iremos usar o método removeFirst para remover e retornar sempre o primeiro elemento da lista, dessa forma, podemos caminhar por toda a lista sem índices.

Primeiramente, constatamos que é essencial para o funcionamento do algoritmo que devemos armazenar a quantidade de filhos e elementos de cada nodo, presumindo que não iremos acessar um elemento mais de uma vez, pois essas são as informações que ditam a estrutura da árvore.

Agora, para simplificarmos a idealização da solução, decidimos focar em um teste de cada vez, começando pela quantidade de nodos da XTree. Para isso, podemos abstrair a quantidade de elementos por nodo, bastando somente identificar a quantidade de filhos que cada nodo possuí. Após isso, fazemos um loop com a quantidade de filhos do nodo, incrementando o total de nodos da árvore a cada passada. Entretanto, existem filhos que podem ter outros filhos, logo devemos chamar recursivamente o mesmo método a cada filho para garantir que verificamos todos os nodos.

Este algoritmo pode funcionar de forma recursiva, da seguinte forma:

```
1       registro list[n] {
2       int soma , int qtdNodos
3       }
4
5       procedimento ANALISAXTREE()
6            // Acessamos os elementos da lista removendo e retornando sempre o primeiro elemento
7            qtdFilhos ← list[0]
8            qtdElementos ← list[0]
9            para i ← 0 até qtdFilhos faça
10                qtdNodos ← qtdNodos + 1
11                ANALISAXTREE()
12           fim
13           para i ← 0 até qtdElementos faça
14                // Novamente só acessamos o primeiro elemento da lista porque vamos removendo
15                soma ← soma + list[0]
16           fim
17      fim
```

Para somarmos todos os elementos contidos nos nodos, devemos abstrair o número de filhos dos nodos para focarmos somente na quantidade de elementos a serem armazenados. Desta maneira, utilizamos um segundo loop para somar todos os valores de um nodo quando ele não tiver mais nenhum filho, desta forma, respeitamos a recursão.

Nesta versão do algoritmo não garantimos a altura da árvore, mas o código pode ser modificado para descobri-la em cima do caminhamento na lista apresentado, basta ir contando cada vez que um filho tem filho e comparar com a maior altura, porém não conseguimos implementar.

Além do método de análise principal, foi desenvolvido também um simples menu para que o usuário escolha qual caso queira testar. Esse menu chama outro método chamado caseTest, (passando o nome do arquivo com o caso escolhido) que por sua vez, administra o teste, onde primeiramente é consumido o arquivo de texto para a lista encadeada, e após isso, calcula o tempo de execução do método que analisa a XTree em milissegundos.

Resultados

Depois de implementar o algoritmo acima em linguagem Java, os casos foram rodados cinco vezes para tirar a média do tempo de execução. Segue abaixo os resultados:

CASO	TAMANHO DO CASO	QUANT. NODOS	SOMA TOTAL	TEMPO (milissegundos)
casom3.txt	3	1	33	0
casom5.txt	4	1	116	0
casom7.txt	864	141	28871	0
casom9.txt	11401	1649	402264	0,6
casom10.txt	76390	10179	2777841	4,8
casom11.txt	447942	56032	16632623	12,2
casom12.txt	2591314	305008	98060765	31,6
casom13.txt	8795004	976719	338643877	55,4

Conclusões

A solução inicial, mesmo não oferecendo um resultado, contribuiu bastante para o entendimento do problema e abriu caminho para a solução definitiva. A partir disso, concluímos que geralmente levantar hipóteses para a solução de um problema e tentar achar equívocos nelas é muito mais inteligente do que simplesmente tentar resolver o problema na prática.

A solução definitiva se mostrou bastante simples e clara, embora eu gostaria testar e discutir outras estruturas de dados para consumir os arquivos de texto, como por exemplo vetores, tendo em vista que teria que se incrementar o seu índice para realizar o caminhamento. Além disso, a solução recursiva adotada foi razoavelmente eficiente, possuindo uma notação O(n), pois devemos passar por todos os elementos da lista e o tempo de execução cresce proporcional a n.