Organizando um Baile: Solução do Problema da Organização de uma festa

Erick Grilo
Max Fratane
Matheus Prado
Vitor Santos

1 Introdução

O problema consiste em que o professor Stewart foi contradado para prestar um serviço de consultoria para o presidente de uma determinada empresa que deseja realizar uma festa e nós vamos ajudá-lo. A empresa possui uma hierarquia tal que a relação de hierarquia forma uma árvore cuja raiz é o presidente da empresa. O departamento do RH classificou cada um dos empregados com uma classificação de conviviabilidade (que é um número real). Para a festa ser proveitosa o máximo para todos que forem, o presidente não quer que ambos um funcionário e seu supervisor vão simultaneamente.

Ao professor Stewart, é dada uma árvore onde cada nó da árvore é um funcionário da companhia, que possui um nome e um valor de conviviabilidade, onde o pai desse nó é o seu supervisor imediato. O objetivo é criar um algoritmo que cria uma lista de convidados que maximiza a soma dos valores de conviviabilidade dos convidados.

2 O Algoritmo

A ideia do algoritmo consiste em solucionar o problema para a árvore de funcionários, com dois casos: caso a árvore tenha somente um nó, e caso a árvore tenha mais de um nó. Dessa forna, a partir do enunciado, temos:

$$Baile(tree) = \begin{cases} \text{funcionário}(tree), \text{caso tree não tenha filhos} \\ \text{MAX} \begin{cases} \sum \text{Baile}(t), \forall t \in \text{filho}(tree) \\ \text{conviviabilidade}(raiz) + \sum Baile(t), \forall t \in \text{neto}(tree) \end{cases} \end{cases} \quad (1)$$

Onde o caso base é o caso de *tree* ser uma árvore com um só nó e a relação de recorrência consiste no caso de tree ter filhos: temos que avaliar então, dentre todos os filhos da árvore de entrada, quais serão os nós da árvore que maximizarão o somatório de conviviabilidade, tomando a precaução de não permitir que um empregado e seu supervisor imediato possam ir ao baile. Nesse caso, na primeira linha da parte do MAX, é avaliado o caso dos filhos de tree (caso o nó da árvore atual vá à festa), e na segunda linha, a conviviabilidade do nó imediatamente acima de tree e os filhos dos seus filhos, ou seja, o funcionário do funcionário da empresa. Dessa forma, é evitado que um funcionário e seu supervisor imediato possam ir ao baile. A função funcionário retorna o funcionário de um determinado nó, enquanto a função conviviabiidade retorna o valor de conviviabilidade de um determinado nó.

Da relação de recorrência apresentada acima, temos o seguinte pseudo-código, que ilustra como o algoritmo funciona:

Entrada: tree: uma árvore com os funcionários da empresa
Saída: A lista de convidados tais que a soma dos valores de conviviabilidade de seus elementos seja a máxima possível

```
Resolver(tree):
início
    Baile(tree,lista)
    retorna lista
fim
```

Algoritmo 1: RESOLVER

Onde, a função *resolver* retorna a lista desejada, enquanto a função principal responsável por procurar a sequência de valores que maximizam a soma das conviviabilidades (*baile*) é chamada no seu escopo. Tal função tem seu comportamento descrito a seguir:

Entrada: tree: uma árvore com os funcionários da empresa, Lista: lista de convidados, inicialmente vazia
Saída: O maior valor de conviviabilidade, de acordo com o caso
Baile(tree,lista):
início
    se *tree já foi visitado* **então**
        lista ← append(lista,lista(tree))//junta a lista de entrada com a lista armazenada no nó
        retorna o valor (?) armazenado em tree (isso nao é a mesma coisa que convivabilidade(tree) nao né?);
    se *tree não possui filhos* **então**
        lista ← adicionar funcionario(tree)//funcionário(tree): retorna o funcionário desse nó da árvore.
        $visitada(tree) \leftarrow True$//o nó tree é marcado como visitado
        $listaArmazenada \leftarrow lista(tree)$//armazena lista de tree
        $crArmazenada \leftarrow conviviabilidade(tree)$
        retorna conviviabilidade(tree)//conviviabilidade(tree): retorna o valor de conviviabilidade desse nó da árvore.
    senão
        $A \leftarrow 0.0$
        $B \leftarrow$ conviviabilidade(tree)
        para cada *filho* $x \in$ *filho(tree)* **faça**
            $A \leftarrow A + Baile(x, listaA)$//listaA: lista auxiliar
            para cada *filho* $y \in$ *filho(x)* **faça**
                $B \leftarrow B + Baile(y, listaB)$//listaB: lista auxiliar
            fim
        fim
        se $B > A$ **então**
            $listaB \leftarrow$ listaB + funcionário(tree)
            $visitada(tree) \leftarrow True$
            $listaArmazenada \leftarrow listaArmazenada + listaB$
            $lista \leftarrow append(lista, listaB)$
            retorna B
        fim
        $visitada(tree) \leftarrow True$
        $listaArmazenada \leftarrow listaArmazenada + listaA$
        $lista \leftarrow append(lista, listaA)$
        $crArmazenada \leftarrow crArmazenada + A$
        retorna A
    fim
fim

Algoritmo 2: Baile