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 convivialidade (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 convivialidade, 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 convivialidade 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:

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

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 convivialidade, 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 convivialidade 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 convivi-abiidade retorna o valor de convivialidade 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 convivialidade 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 convivialidades (*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 convivialidade, de acordo com o caso
Baile(tree,lista):
início
  se *tree já foi visitado* **então**
    $lista \leftarrow$ append(lista,lista(tree))//junta a lista de entrada com a lista armazenada no nó
    retorna o valor armazenado em tree
  $visitada(tree) \leftarrow True$//o nó tree é marcado como visitado
  se *tree não possui filhos* **então**
    $lista \leftarrow$ adicionar funcionario(tree)//funcionário(tree): retorna o funcionário desse nó da árvore.
    $listaArmazenada \leftarrow lista(tree)$//armazena lista de tree
    $crArmazenada \leftarrow convivialidade(tree)$
    retorna convivialidade(tree)//convivialidade(tree): retorna o valor de convivialidade desse nó da árvore.
  senão
    $A \leftarrow 0.0$
    $B \leftarrow$ convivialidade(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)
      $listaArmazenada \leftarrow listaArmazenada + listaB$
      $lista \leftarrow append(lista, listaB)$
      retorna B
    fim
    $listaArmazenada \leftarrow listaArmazenada + listaA$
    $lista \leftarrow append(lista, listaA)$
    $crArmazenada \leftarrow crArmazenada + A$
    retorna A
  fim
fim

Algoritmo 2: BAILE

3 Exemplo

Suponta que a empresa em questão seja uma pequena empresa de informática, e como estamos em junho, A empresa pediu para usarmos o algoritmo para resolvermos o problema em questão referente à uma festa junina. Dessa forma, se um funcionário for, nem seu subordinado imediato nem o seu superior imediato poderão ir. A árvore abaixo ilustra a organização dos funcionários dessa empresa:

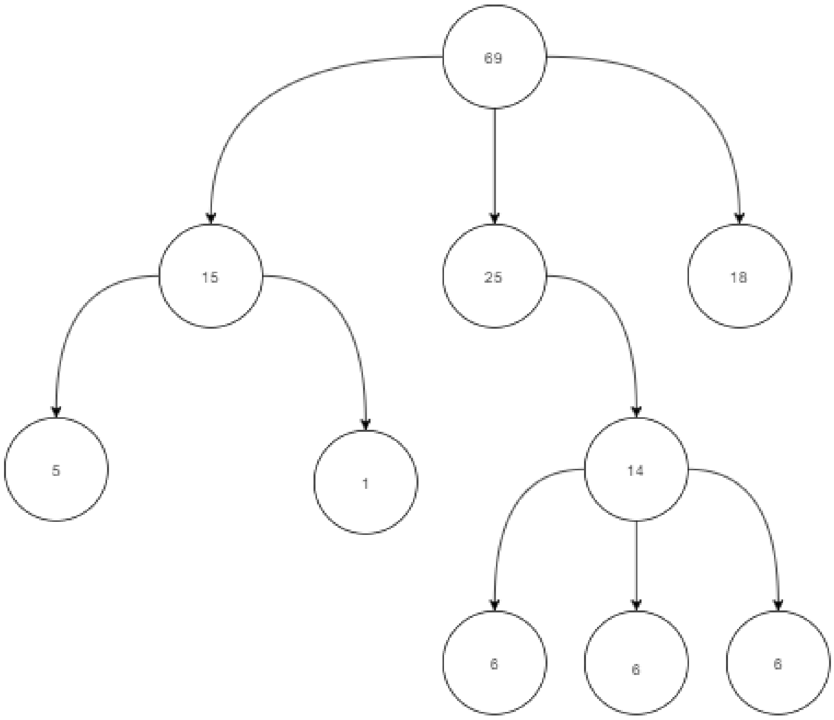

Figura 1: Exemplo de uma árvore que representa funcionários de uma empresa.

Ao se executar o algoritmo, temos que o mesmo verificará que a árvore de entrada não é uma árvore sem filhos. Logo, ele segue o segundo caso descrito na relação de recorrência, que verifica o máximo dos valores entre todos os filhos de *tree* ou o valor de convivialidade da raiz com o dos netos (assim, respeitando a restrição).

Ao entrar no caso de *tree* (a raiz) ter filhos, para cada filho de *tree*, a função *baile* é chamada recursivamente a fim de armazenar o valor de convivialidade de cada um dos filhos de *tree* em uma variável A qualquer. Em seguida, o mesmo ocorre, só que agora para os netos de *tree*, armazenando a soma dos valores em B (que já recebera o valor de convivialidade de *tree*)

4 Prova da Corretude do Algoritmo

A função que resolve o problema é dada pelo pseudocódigo definido em 1 (*resolver*). Porém, a função que efetivamente resolve e processa os dados para solucionar o problema é a 2 (*Baile)*. Dessa forma, ao provarmos a corretude da função *Baile*, provamos a corretude de toda o algortimo que soluciona o problema.

Seja *Baile(T)*: A chamada da função *Baile* tal como descrita na relação de recorrência na seção 2 ou seja, *Baile*(T) retorna o valor máximo do somatório das convivialidades dos seus nós. A prova será feita por indução forte estrutural na árvore de entrada para o problema.

(i) Base da indução: *Baile*(T_0) é verdadeira?
Seja *Baile*(T_0) : O valor máximo da soma de convivialidade da árvore que possui apenas um nó (sem filhos). Como só existe um único nó a ser avaliado, temos que o seu valor de convivialidade (por definição) é o que maximiza o somatório dos valores de convivialidade da árvore. Portanto, a lista retornada por *Resolver* só possui o funcionário representado por esse nó como um convidado. Logo, temos que *Baile*(T_0) vale.

(ii) Hipótese de Indução: Sejam $T_1, T_2, T_3, \ldots, T_n$ sub-árvores formadas a partir da raiz (árvore de entrada para o problema) cujas raízes estão no k-ésimo nível, para k = 1, 2, ..., n. Logo, supor que *Baile*(T_k) vale, para k = 1, 2, ..., n. Por causa dessa suposição, temos que a relação de recorrência também passa a valer para sub-árvores de T_k, ou seja, árvores T_{k_q} que são sub-árvores de uma T_k, com q = 1,2,...,m.

(iii) Passo Indutivo: Provar válido *Baile*($T_n + 1$). Pela hipótese de indução, temos que *Baile*(T_k) é o valor máximo da convivialidade para a sub-árvore de $T_n + 1$, T_k, para k = 1,2,...,n. Dessa forma, temos que *Baile*(T_k) é o valor máximo do somatório do valor de convivialidade de T_k e, portanto, a relação de recorrência encontrada na seção 2 vale. Logo, temos que analisar dois casos:

I) A árvore $T_n + 1$ não possui filhos : pela base da indução, nota-se que o valor de convivalidade máximo de $T_n + 1$ é o seu próprio valor de convivialidade.
II)A árvore $T_n + 1$ possui filhos T_k, k = 1, 2, ..., n,:

Seja $n_1 = \sum Baile(t), \forall t \in$ filho($T_n + 1$). Pela hipótese de indução, temos que *Baile*(T_k) vale. Portanto, n1 é o somatório de todos os valores máximos de convivialidade da árvore $T_n + 1$, para k = 1,2,...,n.
Agora, seja n_2 = convivialidade(raiz) + $\sum Baile(t), \forall t \in$ neto(T_n). Pela hipótese de indução, temos que *Baile*(T_{k_q}) também vale, $\forall$(T_k) sub-árvore da (também sub-árvore) T_k. Tomando $T_k = T_n$, a hipótese de indução vale também para um t neto de T_n.Dessa forma, temos que n2 é o somatório dos valores de convivialidade máximo da árvore T_{k_q}, para k = 1,2,...,n e q = 1,2,...,m.
Logo, por n_1 e n_2, o valor de convivialidade máximo da árvore original que foi dada como entrada para o problema $T_n + 1$ pode ser encontrada pela seguinte fórmula:

$$\textbf{convivialidadeMaxima}(\mathbf{T}_n + 1) = \textbf{MAX(n1,n2).}$$

Temos que a função acima culmina justamente na relação de recorrência que foi encontrada. Como MAX retorna o valor máximo entre dois valores, temos que convivialidadeMaxima retorna o valor máximo dentre os possíveis valores máximos que $T_n + 1$ pode ter. Logo, *Baile*($T_n + 1$) vale.