Redução do problema Lowest Common Ancestor para o problema Range Minimum Query

Vinicius Teixeira de Melo[1], **Wladimir Araújo Tavares**[1]

[1]Universidade Federal do Ceará - Campus Quixadá
CEP 63900-000 – Quixadá – CE – Brasil

viniciusteix@alu.ufc.br, wladimirufc@gmail.com

Abstract. *This article presents the problems Lowest Common Ancestor (LCA) and Range Minimum Query (RMQ), showing the foundations, definitions and some ways to solve those problems. Finally, demonstrates how is the LCA problem reduction for the RMQ.*

Resumo. *Este artigo apresenta os problemas Lowest Common Ancestor (LCA) e Range Minimum Query (RMQ), mostrando os fundamentos, as definições e algumas maneiras de resolver esses problemas. Por fim, demonstra como é feita a redução do problema LCA para o problema RMQ.*

1. Introdução

O problema *Lowest Common Ancestor (LCA)* foi definido por *Alfred Aho, John Hopcroft e Jeffrey Ullman* em [Aho et al. 1976]. Dado dois vértices u e v de uma árvore enraizada T, o problema consiste em encontrar um ancestral em comum de u e v mais distante da raiz. Uma aplicação do *LCA* ocorre em hierarquias de classes em orientação a objetos [Aït-Kaci et al. 1989] permitindo a solução de problemas de controle de herança e polimorfismo. Uma outra aplicação acontece em *Sistemas Distribuídos* [Bender et al. 2005].

Em [Harel and Tarjan 1984], os autores apresentaram um algoritmo com complexidade linear de tempo para preprocessar a árvore T e construir uma estrutura de dados que permitisse que perguntas LCA fossem resolvidas em tempo constante. Vários outros algoritmos mais simples foram apresentados mantendo as mesmas propriedades [Schieber and Vishkin 1988]

Já o problema *Range Minimum Query (RMQ)*, definido em [Berkman and Vishkin 1993], consiste em determinar pelo menor valor em um intervalo de um vetor não-ordenado. Duas das aplicações mais comuns do algoritmo *RMQ* são: determinar o prefixo comum mais longo entre palavras e buscar o ancestral comum mais baixo de dois nós em uma árvore. Porém, na última aplicação é necessário realizar uma redução do problema LCA para o problema RMQ.

Neste artigo será mostrado como o algoritmo *RMQ* pode ser utilizado para determinar o ancestral comum mais baixo de dois nós em uma árvore. Antes disso será apresentado fundamentos necessários para o entendimento dos problemas, a definição de cada problema e, por fim, a redução do algoritmo *LCA* para *RMQ*.

2. Lowest Common Ancestor

Um grafo G é uma estrutura composta por dois conjuntos finitos: (i) um conjunto de vértices, denotado por $V(G)$ e; (ii) um conjunto de arestas, denotado por $E(G)$. Em geral, um grafo G é representado como $G(V, E)$. Uma árvore é um grafo conexo e acíclico.

Em Teoria dos Grafos, o *Lowest Common Ancestor* (LCA) de dois nós u e v de uma árvore é o nó mais baixo que possui u e v como descendentes. Assim, o LCA de u e v é o ancestral de u e v que está mais distante da raiz. Em uma árvore em que cada nó aponta para o seu pai, o ancestral comum mais baixo pode ser facilmente determinado ao encontrar a primeira intersecção dos caminhos de u e v até a raiz, como mostrado a seguir na Definição 1.

Definição 1 *(Funcão LCA(u,v)) Dado dois nós u e v de uma árvore T, a função LCA(u,v) retorna o ancestral x comum mais baixo de u e $v \in T$, tal que x é encontrado por uma busca regressiva a partir dos nós u e v.*

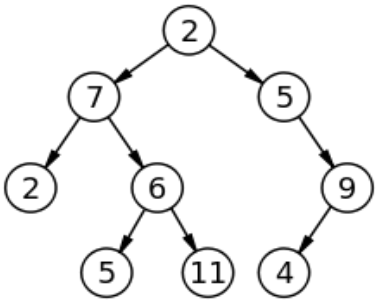

Figura 1. Exemplo de Árvore.

Considere a árvore representada na Figura 1, alguns exemplos de aplicação da funcão $LCA(u, v)$ podem ser vistos a seguir: $LCA(2, 5) = 7$, $LCA(5, 11) = 6$ e $LCA(2, 4) = 2$.

2.1. Solução Trivial

Considere uma árvore T e dois nós u e $v \in T$. Uma solução simples para o problema LCA é dividida em duas etapas: (i) armazena em uma estrutura E todos os nós que estão entre o nó u e a $raiz$ e; (ii) a partir do nó v, também percorre até a raiz, porém, a cada nó v_i visitado, é verificado se $v_i \in E$, se pertencer, $LCA(u, v) = v_i$. A solução trivial mostrada a seguir possui a complexidade $O(N)$.

Algoritmo 1: LCA-TRIVIAL

Entrada: $T, pai[], visitado[], u, v$
Saída: Ancestral comum mais baixo de u e v

```
1 início
2     visitado[i] ← falso, para todo i ∈ T
3     enquanto u != raiz faça
4         visitado[u] ← verdadeiro
5         u ← pai[u]
6     enquanto v != raiz faça
7         se visitado[v] == verdadeiro então
8             retorna v
9         v ← pai[v]
```

2.2. Solução Utilizando Tabela Esparsa

Neste método é necessário fazer um pré-processamento para que as consultas do ancestral comum mais baixo de dois nós seja mais eficiente. Este pré-processamento consiste em calcular todos os ancestrais válidos de cada nó, assim, é necessário uma tabela esparsa para armazenar essas informações.

Algoritmo 2: PRÉ-PROCESSAMENTO

```
   Entrada: P[][], parent[]
 1 início
 2 |  para i ← 1 até N faça
 3 |  |  para j = 0 até (1 << j) < N faça
 4 |  |  |  P[i][j] ← −1
 5 |  para i ← 1 até N faça
 6 |  |  P[i][0] ← parent[i]
 7 |  para j = 1 até (1 << j) < N faça
 8 |  |  para i ← 1 até N faça
 9 |  |  |  se P[i][j − 1] != −1 então
10 |  |  |  |  P[i][j] ← P[P[i][j − 1]][j − 1]
```

Complexidade: $O(N \log N)$

Após o pré-processamento, é necessário a função de própria do algoritmo. Iremos assumir que dados dois nós u e v, o nível deles são o mesmo, ou seja, $nivel[u]$ é igual ao $nivel[v]$. Assim, o *LCA* de u e v estariam sempre a mesma distância, digamos k acima deles. Podemos usar a propriedade de que cada número pode ser expresso pela soma de potências distintas de 2. Portanto, a tabela P é usada para manter a diferença de potências de dois nós, a fim de encontrar o *LCA*. A complexidade do pré-processamento é $O(N \log N)$ e a do $LCA(u, v)$ é $O(\log N)$.

3. Range Minimum Query

O problema *Range Minimum Query* pode ser apresentado da seguinte maneira: dado um vetor não ordenado estático A e dois índices $i \leq j$, qual o menor valor pertencente ao intervalo $A[i], A[i + 1], ..., A[j - 1], A[j]$?. Uma solução simples para esse problema é para cada consulta $RMQ(i, j)$, pesquisar por todo intervalo e devolver o menor valor encontrado, assim, a consulta teria complexidade $O(N)$.

Uma implementação eficiente para esse problema utiliza uma estrutura de dados chamada Árvore de Segmentos (no inglês *Segment Tree*), que é uma estrutura de dados de árvore que armazena informações sobre intervalos tornando a consulta por intervalos mais eficiente. A árvore de segmentos pode ser construída em $O(n)$ e cada consulta $RMQ(i, j)$ pode ser respondida em $O(\log N)$.

4. Redução de LCA para RMQ

Para reduzir o problema *LCA* para *RMQ* é preciso executar o algoritmo de *Euler tour* [Tarjan and Vishkin 1984] na árvore para a obtenção de uma nova representação da mesma. Dada uma árvore T, o algoritmo de *Euler tour* enumera os nós em ordem crescente de acordo com o nível e devolve uma sequência de números, obtidos depois da execução da busca em profundidade que mostra cada vértice visitado.

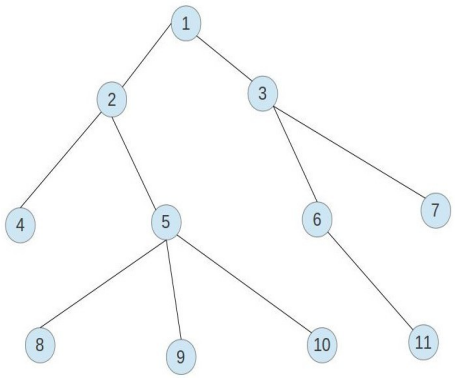

Figura 2. Exemplo de Árvore depois de Enumerada por Nível.

A sequência obtida após a execução do algoritmo de *Euler tour* na árvore da Figura 2 é: $[1, 2, 4, 2, 5, 8, 5, 9, 5, 10, 5, 2, 1, 3, 6, 11, 6, 3, 7, 3, 1]$.

Considere $F[no]$ como o índice da primeira ocorrência do nó no vetor E obtido após a execução do algoritmo de *Euler tour*. Seguindo de $E[F[u]]$ até $E[F[v]]$, são encontrados todos os nós que estão no caminho de u até v. Assim, o $RMQ(F[u], F[v])$ retorna o índice de um elemento no vetor de *Euler tour*, de modo que esteja entre $F[u]$ e $F[v]$ e seja o mais baixo possível.

Algoritmo 3: REDUÇÃO-LCA-RMQ

Entrada: $E[], F[], u, v$
Saída: Ancestral comum mais baixo de u e v
1 **início**
2 | **retorna** $E[RMQ(F[u], F[v])]$

Tabela 1. Comparação das Complexidades do *RMQ*

Solução	Pré-processamento	Consulta
Tabela Esparsa	O(N log N)	O(1)
Segment Tree	O(N)	O(log N)

5. Conclusão

Este artigo apresentou um estudo sobre redução do problema *Lowest Common Ancestor* para o problema *Range Minimum Query*, desde o estudo da arte de cada algoritmo, os fundamentos, as definições de cada problema, até o algoritmo de redução. Portanto, podemos ver que a complexidade que interfere no algoritmo de redução é a complexidade do algoritmo de descobrir o menor valor em um dado intervalo.

Referências

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1976). On finding lowest common ancestors in trees. *SIAM Journal on computing*, 5(1):115–132.

Aït-Kaci, H., Boyer, R., Lincoln, P., and Nasr, R. (1989). Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(1):115–146.

Bender, M. A., Farach-Colton, M., Pemmasani, G., Skiena, S., and Sumazin, P. (2005). Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94.

Berkman, O. and Vishkin, U. (1993). Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242.

Harel, D. and Tarjan, R. E. (1984). Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355.

Schieber, B. and Vishkin, U. (1988). On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262.

Tarjan, R. E. and Vishkin, U. (1984). Finding biconnected componemts and computing tree functions in logarithmic parallel time. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 12–20. IEEE.