Árvores

Árvores Binárias de Busca: Balanceamento

Prof. Edson Alves - UnB/FGA

2018

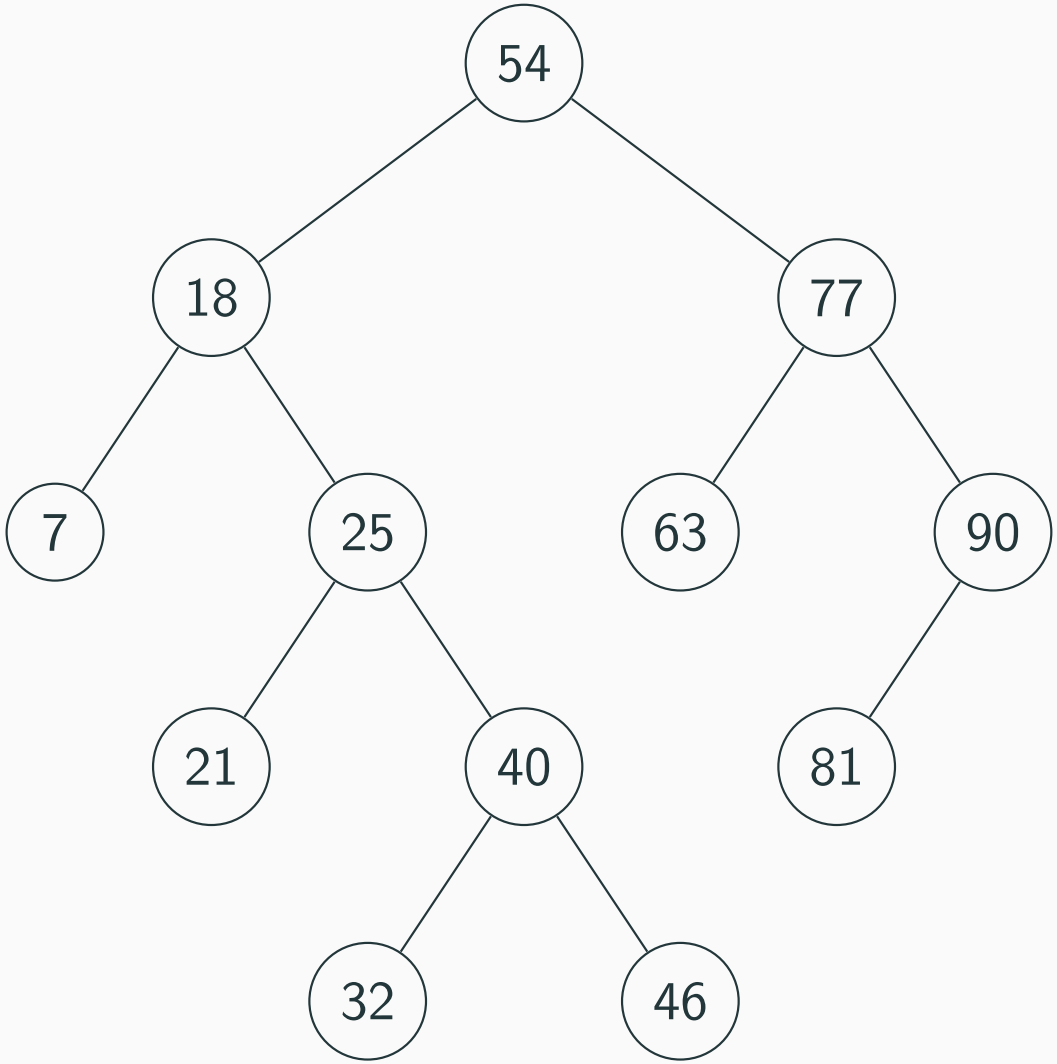
Sumário

Balanceamento de árvores binárias

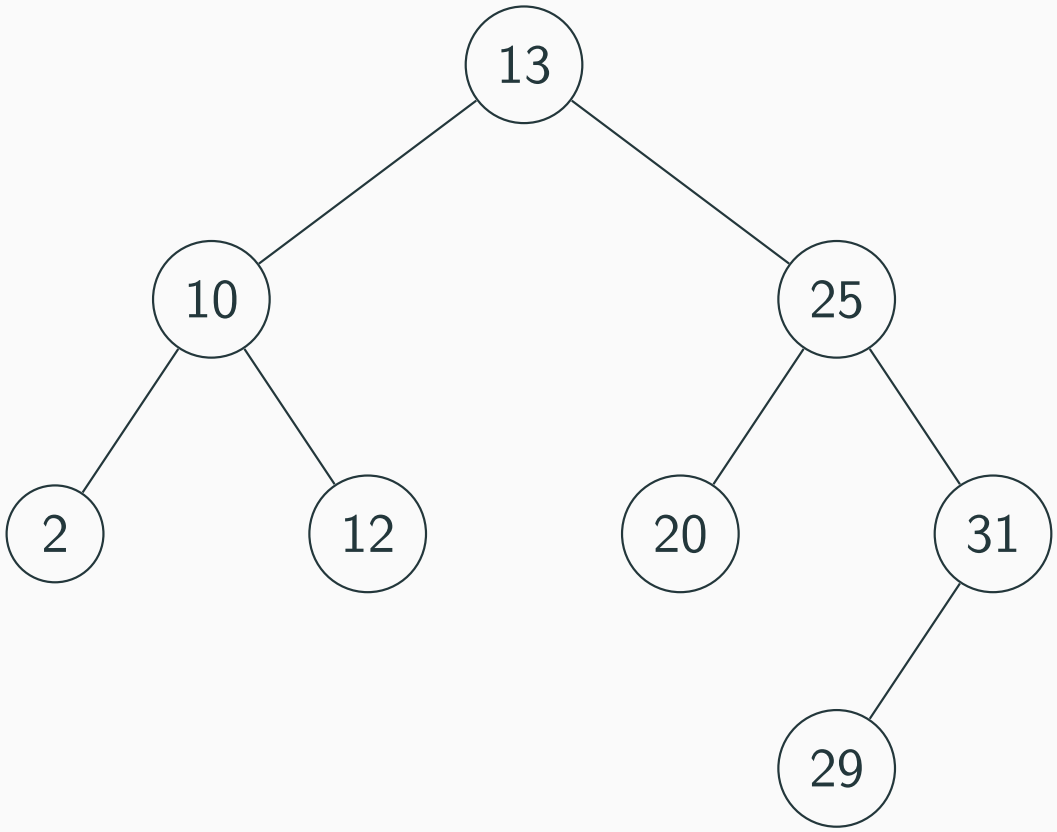
Balanceamento

- Uma árvore binária está balanceada se a diferença de altura das duas subárvores de qualquer nó da árvore é menor ou igual a 1
- Uma árvore binária está perfeitamente balanceada se ela está balanceada e todos os seus nó se encontram em, no máximo, dois níveis distintos
- Uma árvore binária é dita completa se todas as suas folhas estão no mesmo nível
- A altura de uma árvore é igual ao nível máximo dentre todos os nós da árvore
- Uma árvore pode estar balanceada sem estar perfeitamente balanceada

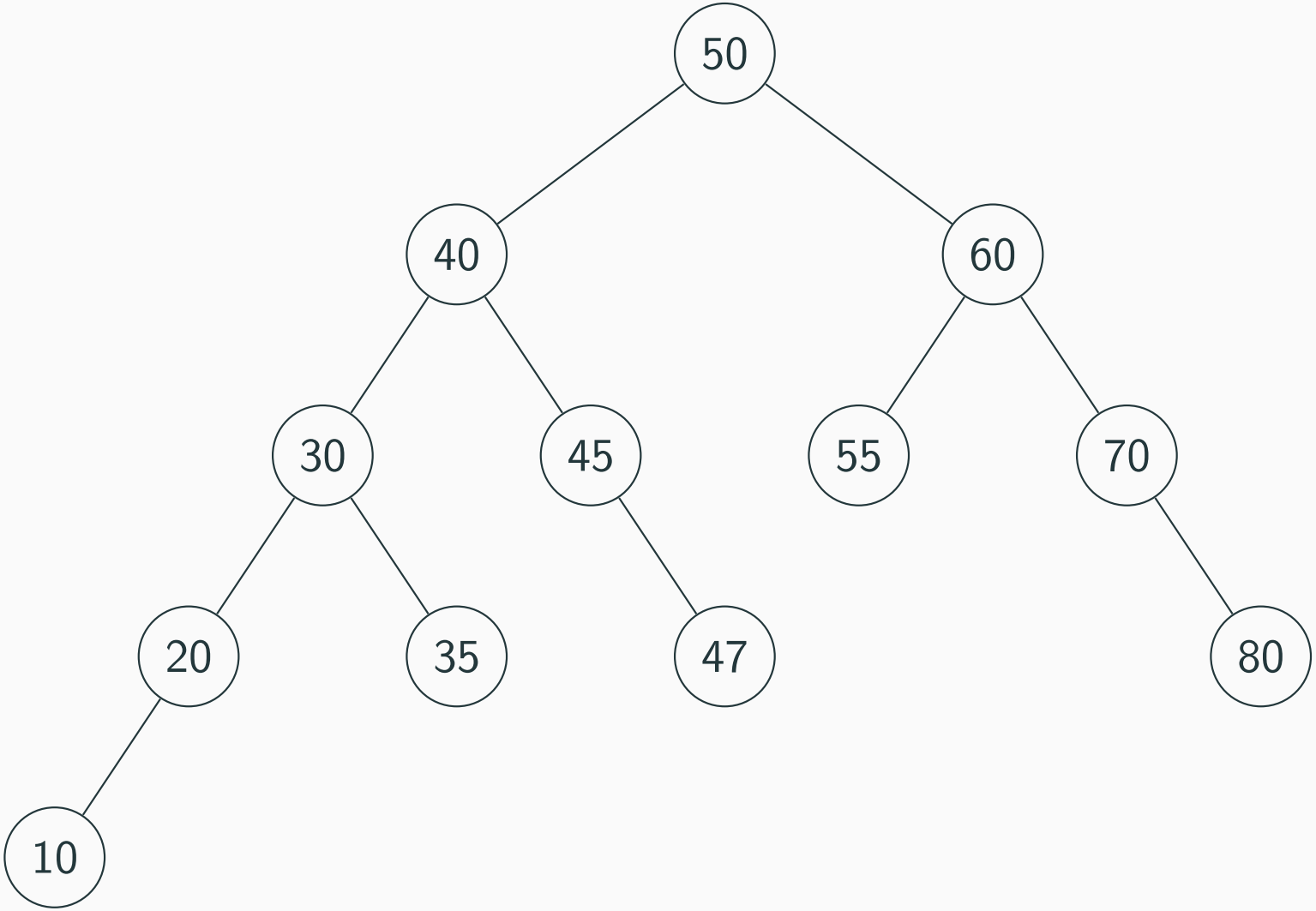
Exemplo de árvore desbalanceada



Exemplo de árvore perfeitamente balanceada



Exemplo de árvore balanceada, mas não perfeitamente balanceada



Algoritmo para verificação de balanceamento

- É possível utilizar a recursão para checar o balanceamento a cada nó, usando também a rotina recursiva de cálculo de tamanho
- O algoritmo é de fácil entendimento, mas possui maior complexidade assintótica $O(N^2)$
- É possível reduzir a complexidade para $O(N)$ através do uso de programação dinâmica
- Para verificar se uma árvore está perfeitamente balanceada, primeiro é necessário verificar se a mesma está balanceada
- Em seguida, determina-se a altura da árvore: se ela não extrapolar o menor inteiro maior do que $\log(N+1)$, a árvore está perfeitamente balanceada

Implementação do algoritmo de verificação de balanceamento

```
1  template<typename T>
2  class BinaryTree {
3  private:
4      struct Node {
5          T info;
6          Node *left, *right;
7      };
8
9      Node *root;
10
11     int heigth(const Node *node) const
12     {
13         if (node == nullptr)
14             return 0;
15
16         return std::max(heigth(node->left), heigth(node->right)) + 1;
17     }
```

Implementação do algoritmo de verificação de balanceamento

```
18
19      bool is_balanced(const Node *node) const
20      {
21          if (node == nullptr)
22              return true;
23
24          int L = heigth(node->left);
25          int R = heigth(node->right);
26
27          if (abs(L - R) > 1)
28              return false;
29
30          return is_balanced(node->left) and is_balanced(node->right);
31      }
32
33  public:
34      BinaryTree() : root(nullptr) {}
35
36      bool is_balanced() const { return is_balanced(root); }
37  };
```

Algoritmos de balanceamento

Balanceamento por inserção

- Uma maneira de garantir o balanceamento de uma árvore binária de busca é utilizar um processo de inserção controlada, de modo que ao final das inserções a árvore resultante esteja balanceada
- Futuras inserções ou remoções podem resultar no desbalanceamento da árvore
- Esta estratégia é útil quando os elementos a serem inseridos são conhecidos de antemão e quando não haverão novas inserções ou remoções
- O algoritmo $O(N \log N)$ que resulta em uma árvore balanceada é
  1. armazene os N elementos a serem inseridos em um vetor v
  2. ordene v em ordem crescente
  3. insira o elemento central x do vetor na árvore
  4. continue o processo no subvetor à esquerda de x
  5. finalize o processo no subvetor à direita de x

Algoritmo de inserção balanceada

```
template<typename T>
class BST {
private:
    struct Node {
        T info;
        Node *left, *right;
    };

    Node *root;

    void BST(BinaryTree& tree, const vector<T>& vs, int a, int b)
    {
        if (a <= b)
        {
            int m = a + (b - a)/2;

            tree.insert(vs[m]);
            balanced_insertion(tree, vs, a, m - 1);
            balanced_insertion(tree, vs, m + 1, b);
        }
    }
```

Algoritmo de inserção balanceada

```
22
23 public:
24     BST() : root(nullptr) {}
25
26     void insert(const T& info);
27
28     static BinaryTree balanced(const std::vector<T>& xs)
29     {
30         std::vector<T> vs(xs);
31         std::sort(vs.begin(), vs.end());
32
33         BST tree;
34         balanced_insertion(tree, vs, 0, vs.size() - 1);
35
36         return tree;
37     }
38 };
```

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C++ Reference[1].

[1] https://en.cppreference.com/w/