


# Árvore 2 3 4

Vitor Barcelos de Cerqueira
Ramon Basto Callado
Daniel melo de lima
Erick Pernambuco

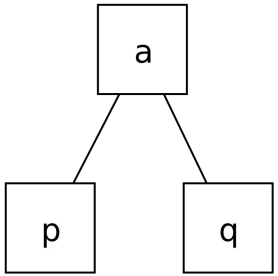
https://github.com/ViBC27/Data-Structures

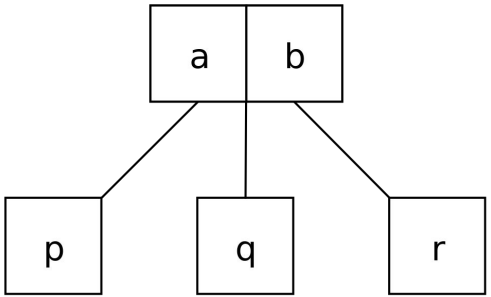
# Motivação

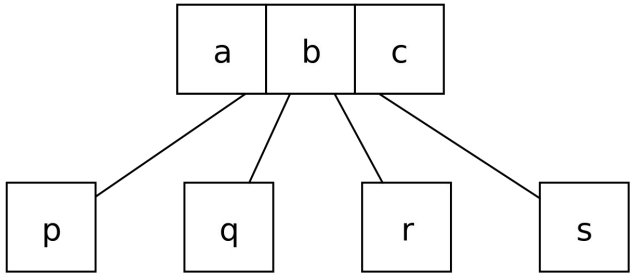
- O balanceamento das AVLs aumenta seu custo computacional;
- Prever palavras ou buscá-la dentro de um dicionário;

# Árvore 2 3 4

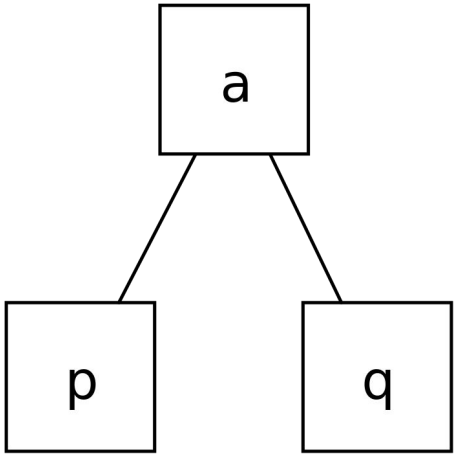
- A 2 3 4 é uma árvore auto-balanceada;
- Todas as folhas possuem a mesma profundidade;
- Possui 3 tipos de nós:
  - 2-nó : um elemento e dois filhos;
  - 3-nó : dois elementos e três filhos;
  - 4-nó : três elementos e quatro filhos;



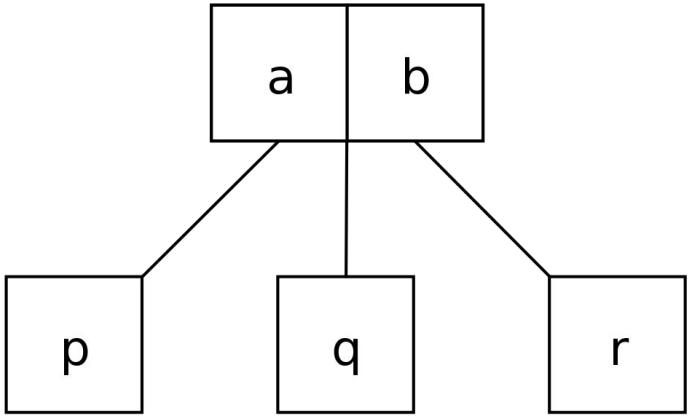



# Propriedade 2-nó



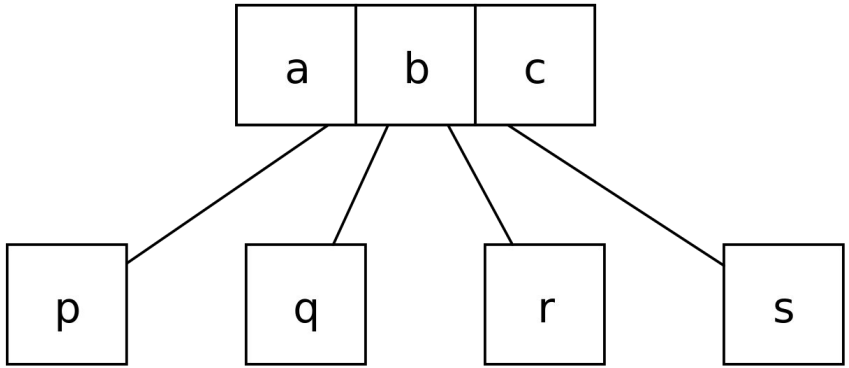

- Todo valor inserido na subárvore *p* é <= *a*;
- Todo valor inserido na subárvore *q* é > *a*;

# Propriedade 3-nó

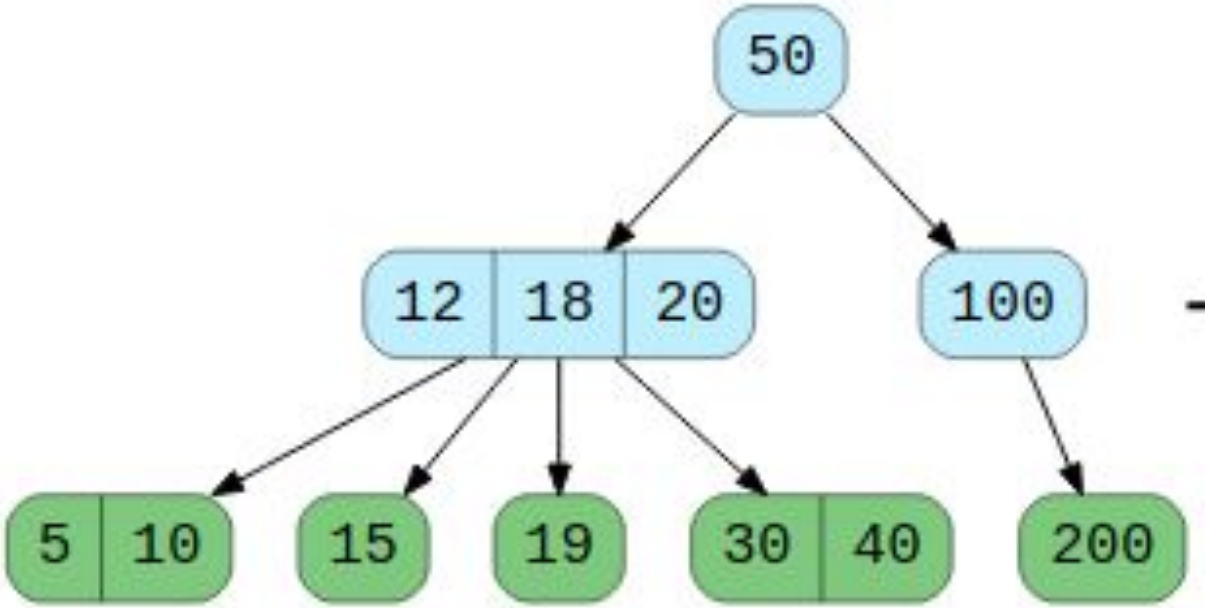



- Todo valor inserido na subárvore *p* é <= *a*;
- Todo valor inserido na subárvore *q* é > *a* e <= *b*;
- Todo valor inserido na subárvore *r* é > *b*;

# Propriedade 4-nó




- Todo valor inserido na subárvore *p* é <= *a*;
- Todo valor inserido na subárvore *q* é > *a* e <= *b*;
- Todo valor inserido na subárvore *r* é > *b* e <= *c*;
- Todo valor inserido na subárvore *s* é > *c*;


50
12
18
20
100
5
10
15
19
30
40
200

# Inserção

- A inserção é feita nas folhas (de baixo para cima );
- Se realiza uma busca por profundidade até achar uma folha;
- Ao realizar a busca é feita uma divisão de todos os nós cheios que encontrarmos.
- No pior caso da inserção temos log n + 1:
    - Quando ocorre em que todos os nós do caminho serem do tipo 4-nó;

# Divisão de um 4-nó

- Escolha o elemento do meio e mova para o pai. Se for a raiz e não tiver pai, uma nova raiz será criada e o elemento do meio será movido para a nova raiz.
- Crie 2 nó do tipo 2-nó e mova o elemento da esquerda para um nó, e o elemento da direita para o outro.
- Se o nó que estava cheio tiver filhos esses filhos serão filhos dos 2 nós tipo 2-nó recém criados.

OBS.: Não haverá divisões em cascata.

# Animação

https://www.educative.io/page/5689413791121408/80001

https://www.youtube.com/watch?v=I0eHbIEGCYo

```
struct tree234{
        int item[3];
        int type;
        tree234 *parent;
        tree234 *childers[4];
};
```

```
int searchTree234(tree234 *tree, int item){

  if(tree == NULL){
    return 0;
  }
//2-nó
  else if(tree->type == 2){
    if(tree->item[0] == item){
      return 1;
    }
    else if(tree->item[0] > item){
      return searchTree234(tree->childers[0], item);
    }
    else{
      return searchTree234(tree->childers[1], item);
    }
  }
//3-nó
  else if(tree->type == 3){
    if(tree->item[0] == item || tree->item[1] == item){
      return 1;
    }
    else if(tree->item[0] > item){
      return searchTree234(tree->childers[0], item);
    }
    else if(tree->item[1] > item){
      return searchTree234(tree->childers[1], item);
    }
    else{
      return searchTree234(tree->childers[2], item);
    }
  }
//4-nó
  else{
    if(tree->item[0] == item || tree->item[1] == item ||
tree->item[2] == item){
      return 1;
    }
    else if(tree->item[0] > item){
      return searchTree234(tree->childers[0], item);
    }
    else if(tree->item[1] > item){
      return searchTree234(tree->childers[1], item);
    }
    else if(tree->item[2] > item){
      return searchTree234(tree->childers[2], item);
    }
    else{
      return searchTree234(tree->childers[3], item);
    }
  }

}
```

# Conclusão

A estrutura atende as necessidades dos dicionários e teclados com auto-complete;

Não tem o custo adicional do balanceamento das AVLs.