

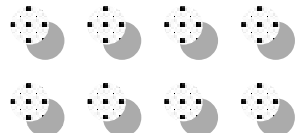
# OBI

## Aula 08

GEMP  
Grupo de Estudos da  
Maratona de Programação

# Tópicos

- O que vimos?
- Árvores Rubro-Negras (intro)
- Map
- Set
- Exemplo
- Exercício



# O que vimos...

- **Manipulações de Vetores**
  - Ordenação
  - Stacks (Pilhas)
  - Filas (Queues)

```
bubble_sort_asc.cpp

#include <iostream>
using namespace std;

void print(int* arr, int tam);

int main() {
    int vet[]{100, 95, 78, 51, 59, 85, 4, 3, 2, 1};

    int tam = sizeof(vet)/sizeof(int);

    cout << "Original Array ⇒ ";
    print(vet, tam);

    // Bubble Sort
    for (int i = 0; i < tam - 1; i++) {
        for (int u = 0; u < tam - 1 - i; u++) {
            if (vet[u] > vet[u+1]) {
                int tmp = vet[u+1];
                vet[u+1] = vet[u];
                vet[u] = tmp;
            }
        }
    }

    cout << "Sorted Array (Crescente/Ascending) ⇒ ";
    print(vet, tam);
    return 0;
}

void print(int* arr, int tam) {
    for (int i = 0; i < tam; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

```
#include <iostream>
using namespace std;

void print(int* arr, int tam);

int main() {
    int vet[]{100, 95, 78, 51, 59, 85, 4, 3, 2, 1};

    int tam = sizeof(vet)/sizeof(int);

    cout << "Original Array ⇒ ";
    print(vet, tam);

    // Bubble Sort
    for (int i = 0; i < tam - 1; i++) {
        for (int u = 0; u < tam - 1 - i; u++) {
            if (vet[u] < vet[u+1]) {
                int tmp = vet[u+1];
                vet[u+1] = vet[u];
                vet[u] = tmp;
            }
        }
    }

    cout << "Sorted Array (Decrescente/Descending) ⇒ ";
    print(vet, tam);
    return 0;
}

void print(int* arr, int tam) {
    for (int i = 0; i < tam; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

```
#include <iostream>
using namespace std;

void print(int* arr, int tam);

int main() {
    int vet[]{100, 95, 78, 51, 59, 85, 4, 3, 2, 1};

    int tam = sizeof(vet)/sizeof(int);

    cout << "Original Array ⇒ ";
    print(vet, tam);

    // Selection sort
    for (int i = 0; i < tam; i++) {
        int menor_val_index = i;
        for (int u = i; u < tam; u++) {
            if (vet[menor_val_index] > vet[u])
                menor_val_index = u;
        }
        int tmp = vet[i];
        vet[i] = vet[menor_val_index];
        vet[menor_val_index] = tmp;
    }

    cout << "Sorted Array (Crescente/Ascending) ⇒ ";
    print(vet, tam);
    return 0;
}

void print(int* arr, int tam) {
    for (int i = 0; i < tam; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

```
selection_sort_desc.cpp

#include <iostream>
using namespace std;

void print(int* arr, int tam);

int main() {
    int vet[]{100, 95, 78, 51, 59, 85, 4, 3, 2, 1};

    int tam = sizeof(vet)/sizeof(int);

    cout << "Original Array ⇒ ";
    print(vet, tam);

    // Selection sort
    for (int i = 0; i < tam; i++) {
        int maior_val_index = i;
        for (int u = i; u < tam; u++) {
            if (vet[maior_val_index] < vet[u])
                maior_val_index = u;
        }
        int tmp = vet[i];
        vet[i] = vet[maior_val_index];
        vet[maior_val_index] = tmp;
    }

    cout << "Sorted Array (Decrescente/Descending) ⇒ ";
    print(vet, tam);
    return 0;
}

void print(int* arr, int tam) {
    for (int i = 0; i < tam; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```



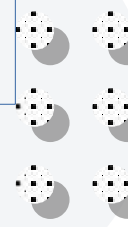

Filas (Queues)



- FIFO (First In First Out)
- Primeiro que entra é o primeiro que sai

Stacks (Pilhas)



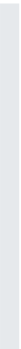
- FILO (First In Last Out)
  - Primeiro que entra é o último que sai
- 
- 



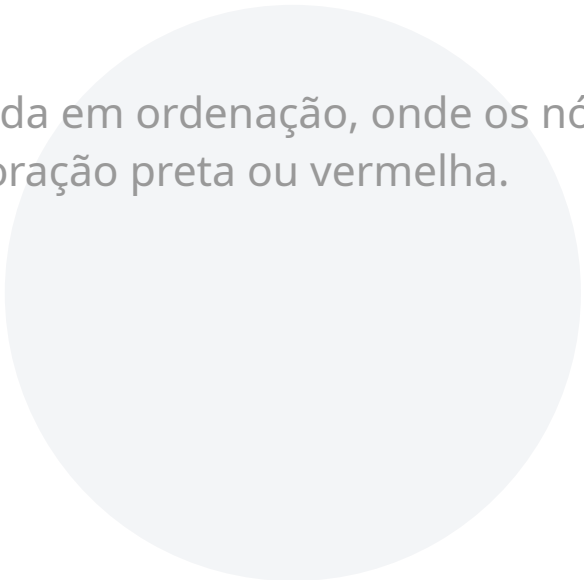


# Árvores Rubro-Negras

## Árvores Rubro-Negras



Estrutura de dados focada em ordenação, onde os nós que a compõe possuem coloração preta ou vermelha.



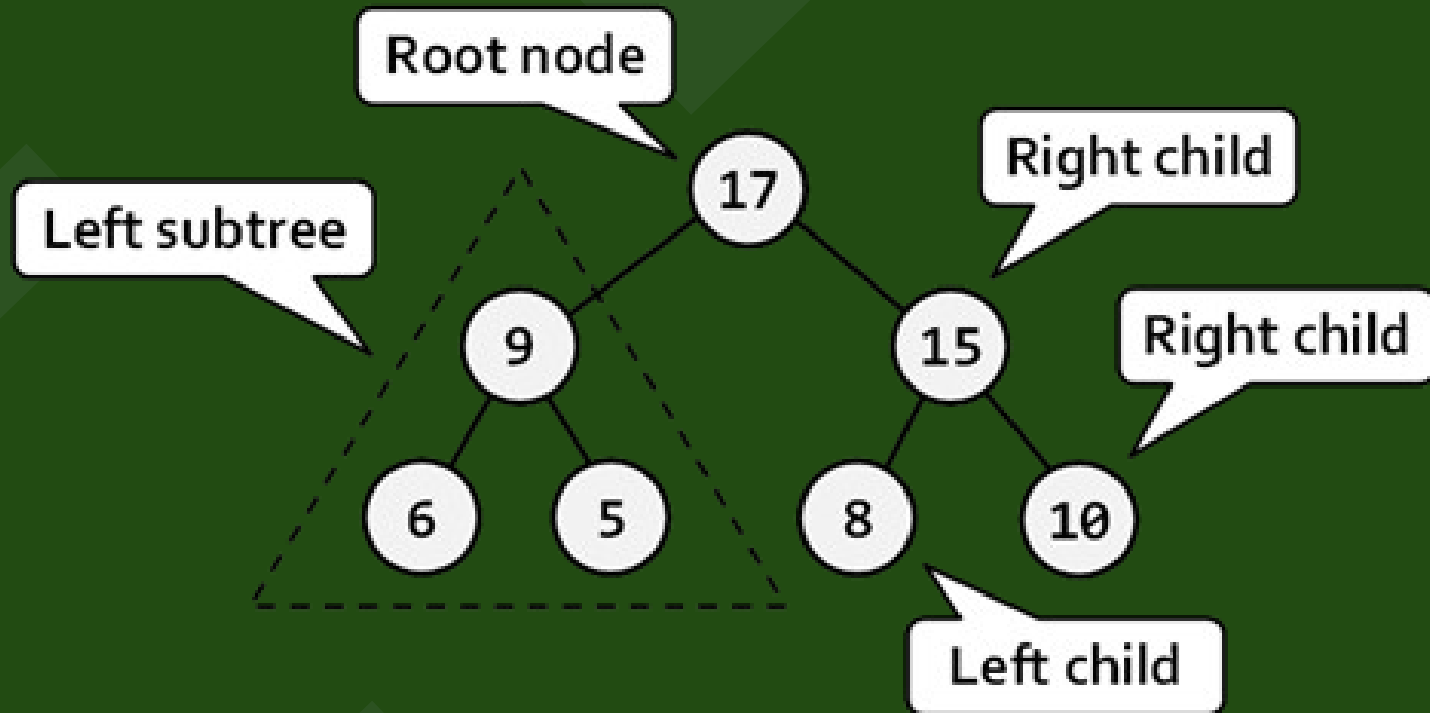
# Árvore Rubro-Negra



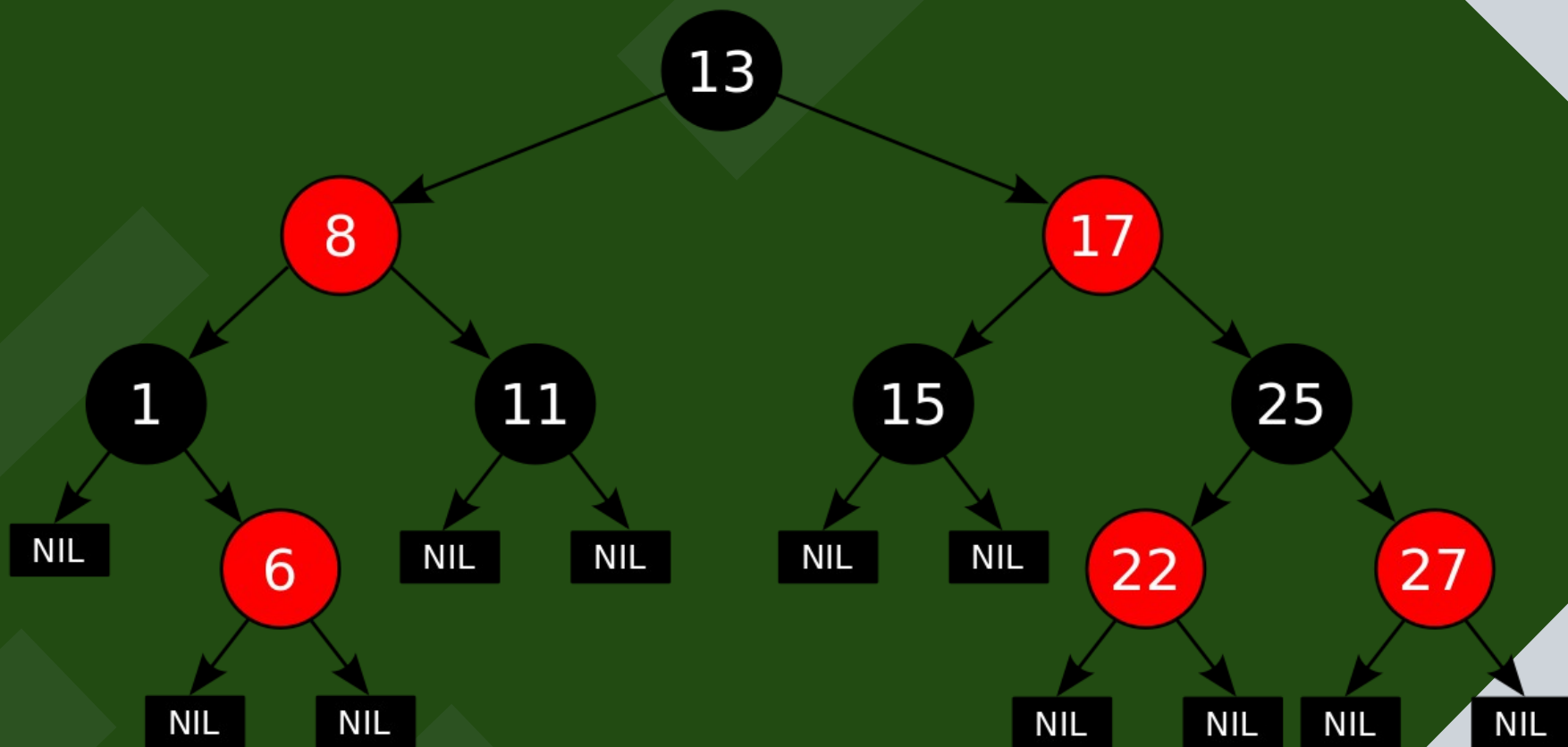
+



# Árvore



# Árvore Rubro-Negra

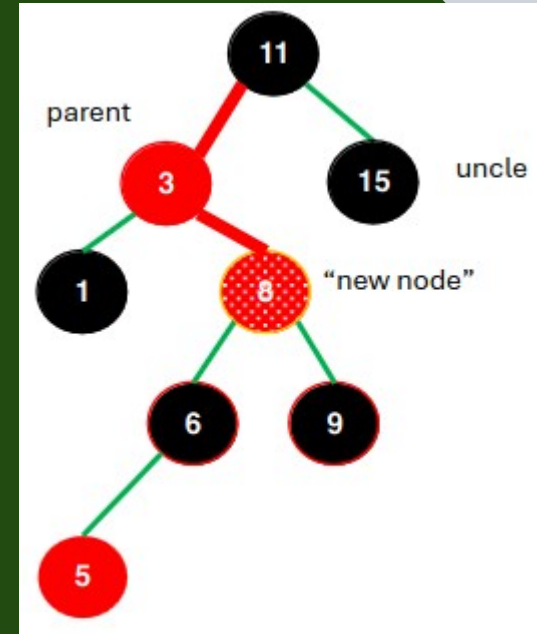
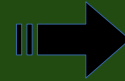
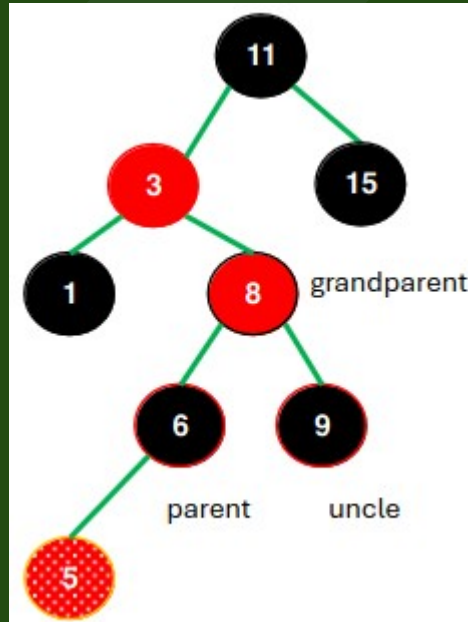
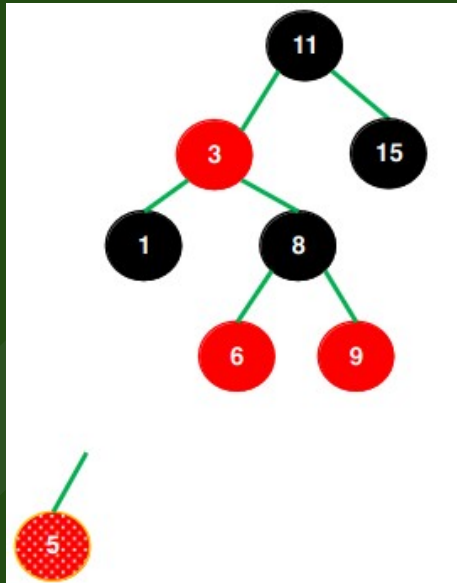


# Propriedades

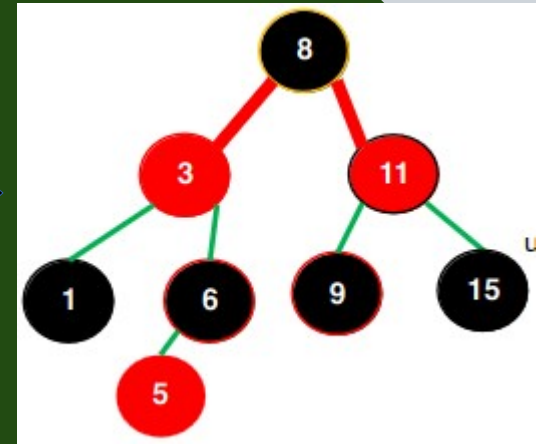
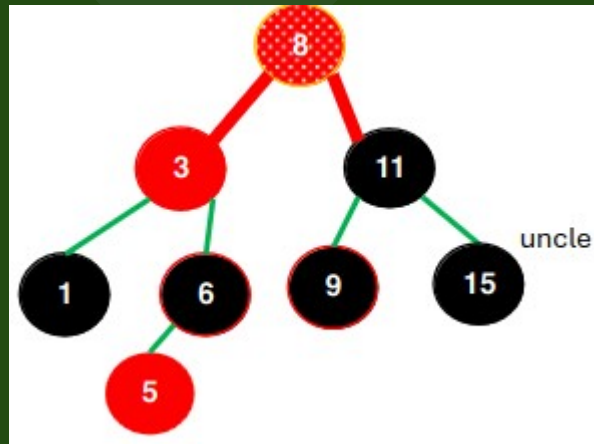
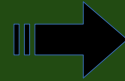
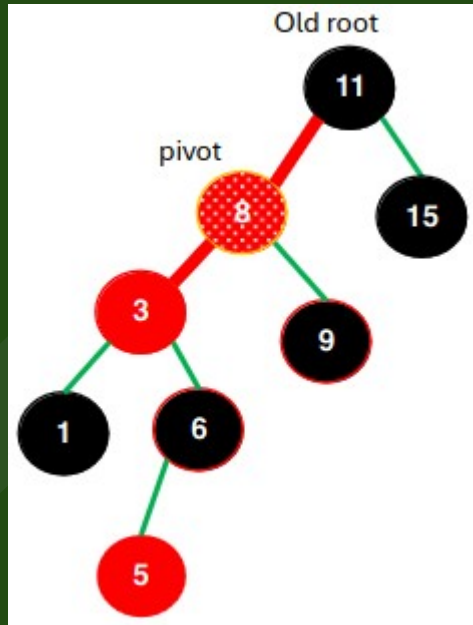
1. Um nó é vermelho ou preto.
2. A raiz é preta. (Esta regra é usada em algumas definições. Como a raiz pode sempre ser alterada de vermelho para preto, mas não sendo válido o oposto, esta regra tem pouco efeito na análise.)
3. Todas as folhas(nil) são pretas.
4. Ambos os filhos de todos os nós vermelhos são pretos.
5. Todo caminho de um dado nó para qualquer de seus nós folhas descendentes contém o mesmo número de nós pretos.



# Exemplo de balanceamento



# Exemplo de balanceamento



# Map

## Map

Estrutura de dados que armazenam dados por pares chave-valor.



# Exemplo

Fazer um programa com os nomes e as idades das pessoas presentes, parar quando houver no lugar do nome a palavra "fim".

Ao final imprimir os nomes em ordem lexical.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> names_and_birthdays{{"Renato", 23}, {"Djalma", 25}};

    while (true) {
        string name;
        cin >> name;
        if (name == "fim")
            break;
        else {
            int age;
            cin >> age;
            names_and_birthdays[name] = age;
        }
    }

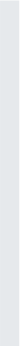
    cout << "\nBy alphabetic order (Asc.): \n\n";
    for (pair<string, int> name_birthday : names_and_birthdays)
        cout << "\t* Name: " << name_birthday.first << ", Age: " << name_birthday.second << endl;
    cout << endl;

    return 0;
}
```

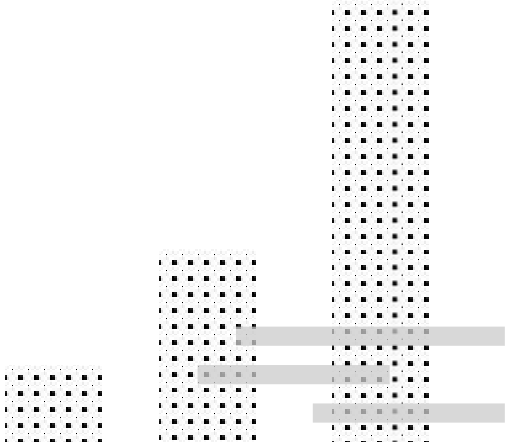



# Set

## Set



Estrutura de dados que armazenam dados de maneira similar a teoria de conjuntos moderna.





# Exemplo

Pegar as idades de todos os presentes e as imprimir em ordem decrescente cada idade única.

