MC-202 — Aula 14
Árvores de Afunilamento

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

Introdução

No nosso aparelho celular, temos diversos contatos.

Introdução

Carlos : 3234 2322
Leo : 92923 2932
José : 92832 3936
Ana : 99129 4323
Maria : 9992 3432
Daniel : 99381 3824
Bia : 3323 3432

No nosso aparelho celular, temos diversos contatos.

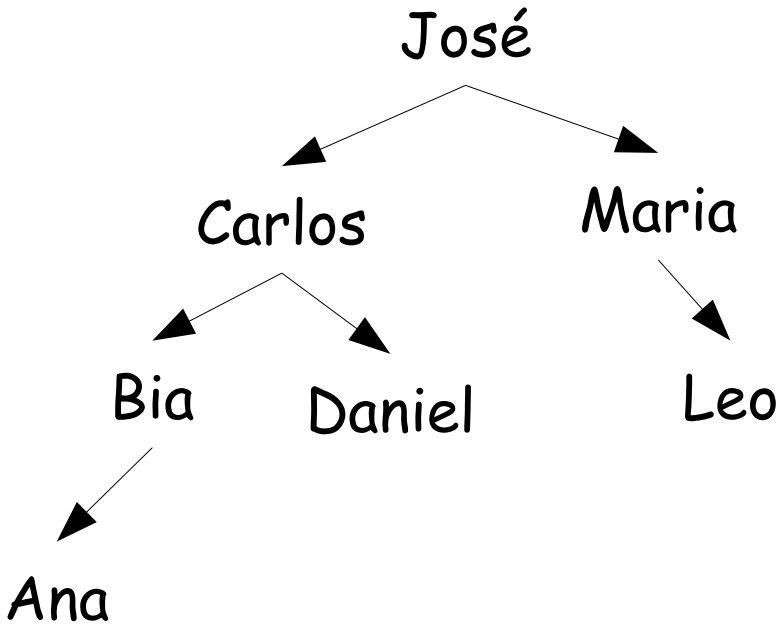
Introdução

Carlos : 3234 2322
Leo : 92923 2932
José : 92832 3936
Ana : 99129 4323
Maria : 9992 3432
Daniel : 99381 3824
Bia : 3323 3432

No nosso aparelho celular, temos diversos contatos.

Pergunta: Que estrutura é a mais adequada para guardar os dados?

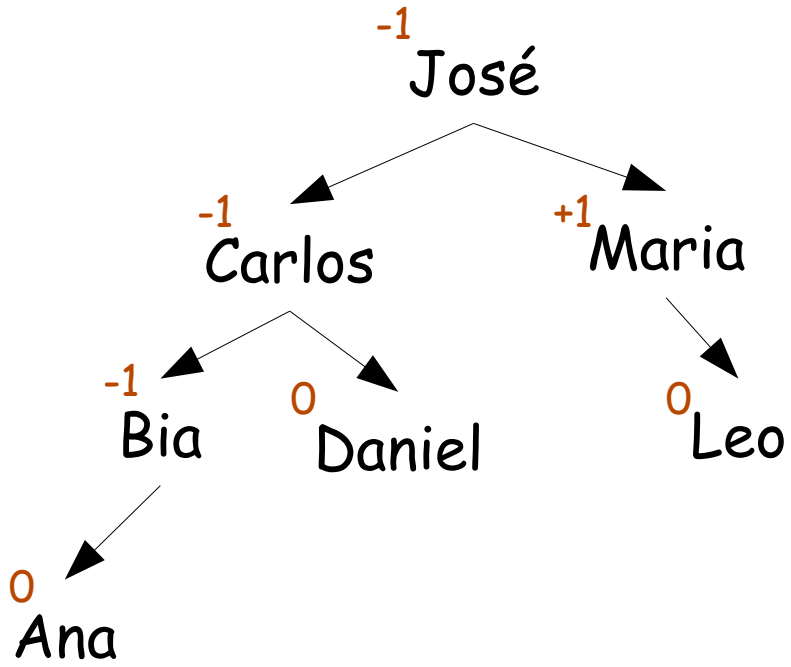
Introdução



No nosso aparelho celular, temos diversos contatos.

Pergunta: Que estrutura é a mais adequada para guardar os dados?
Resposta: Uma árvore

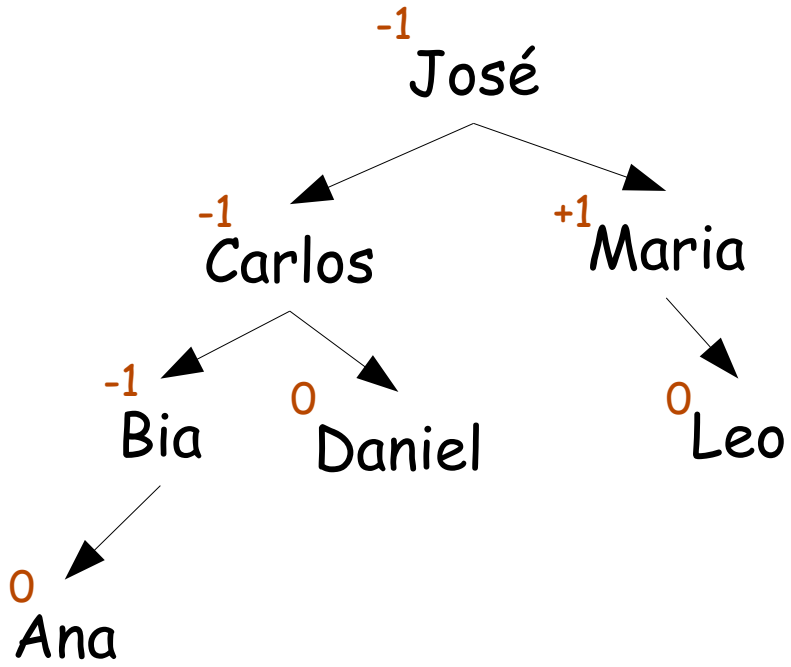
Introdução



No nosso aparelho celular, temos diversos contatos.

Pergunta: Que estrutura é a mais adequada para guardar os dados?
Resposta: Uma árvore balanceada

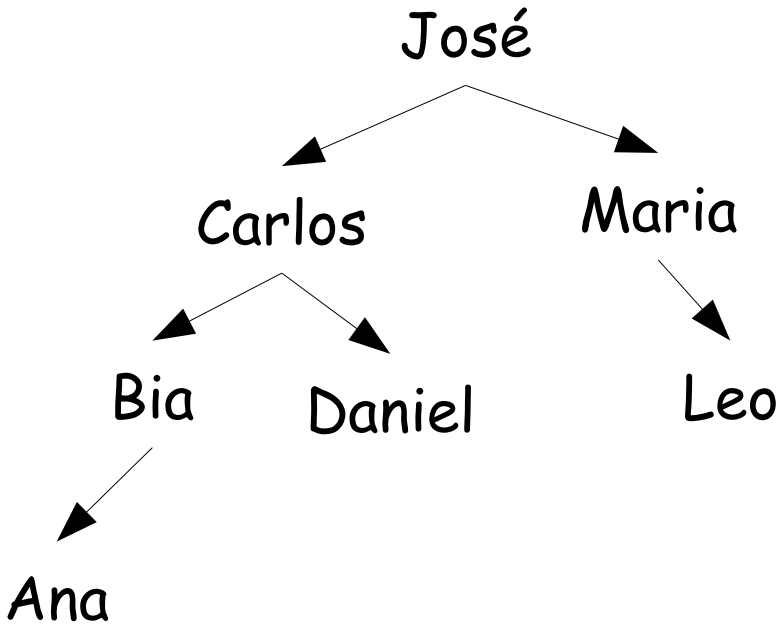
Introdução



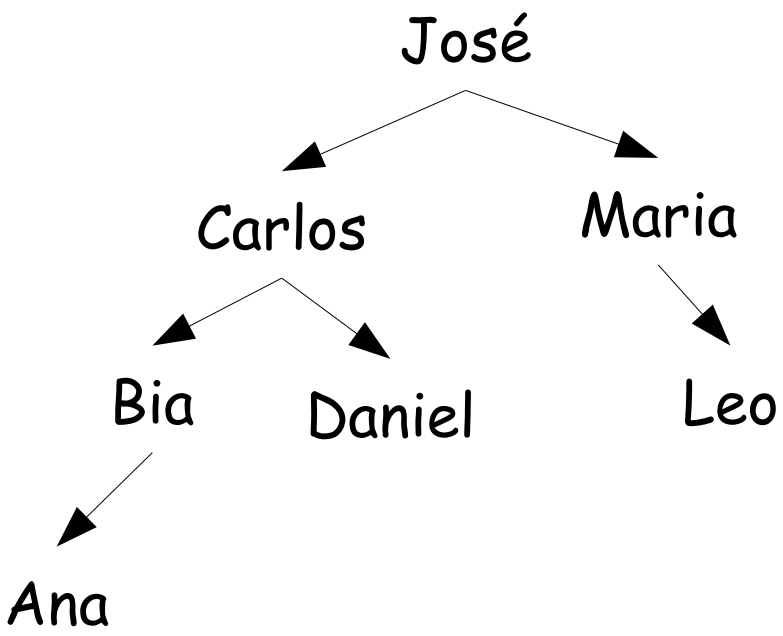
No nosso aparelho celular, temos diversos contatos.

Pergunta: Que estrutura é a mais adequada para guardar os dados?
Resposta: Uma árvore balanceada?

Verificando nossa escolha



Verificando nossa escolha

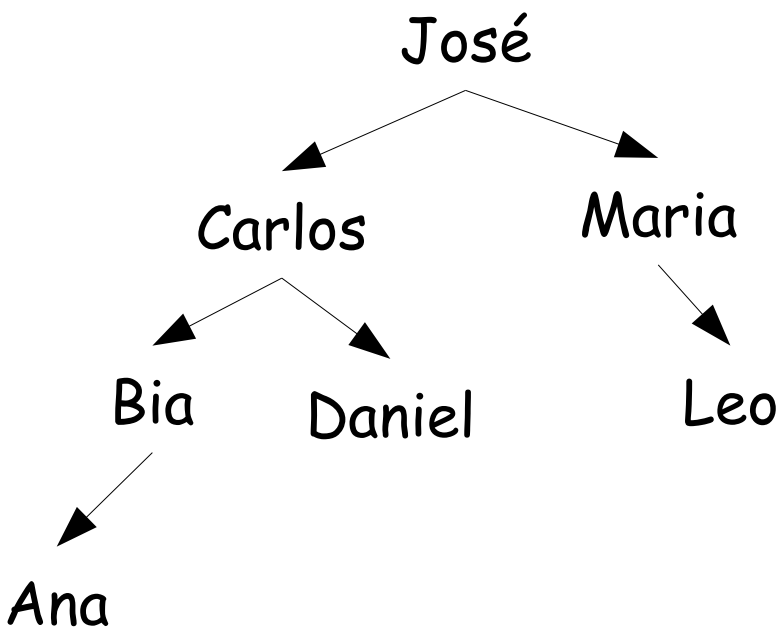


Acessos:

0

Contando acessos

Verificando nossa escolha



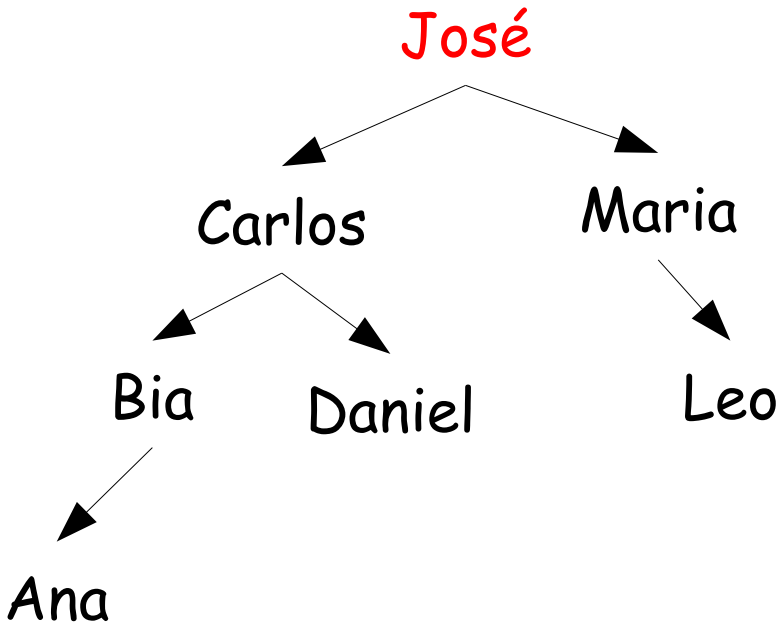
Acessos:

0

Contando acessos

Chamamos Maria

Verificando nossa escolha



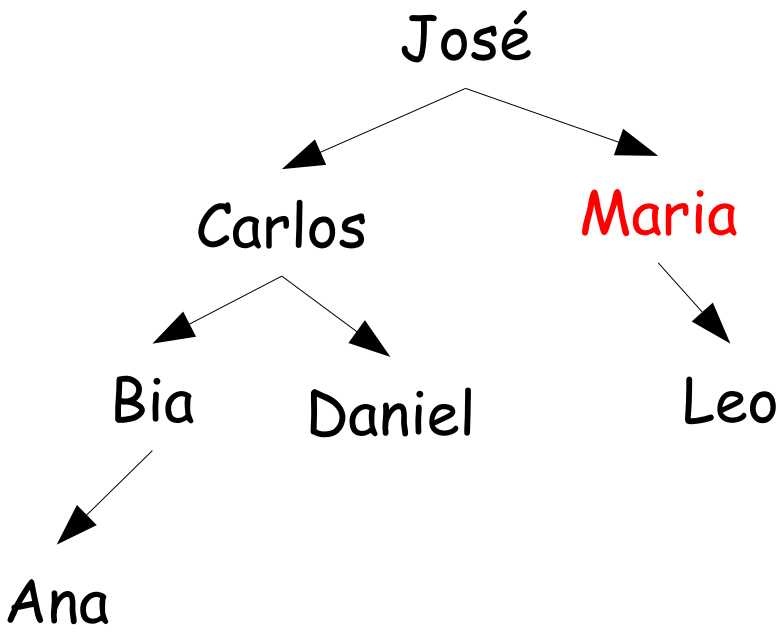
Acessos:

1

Contando acessos

Chamamos Maria

Verificando nossa escolha



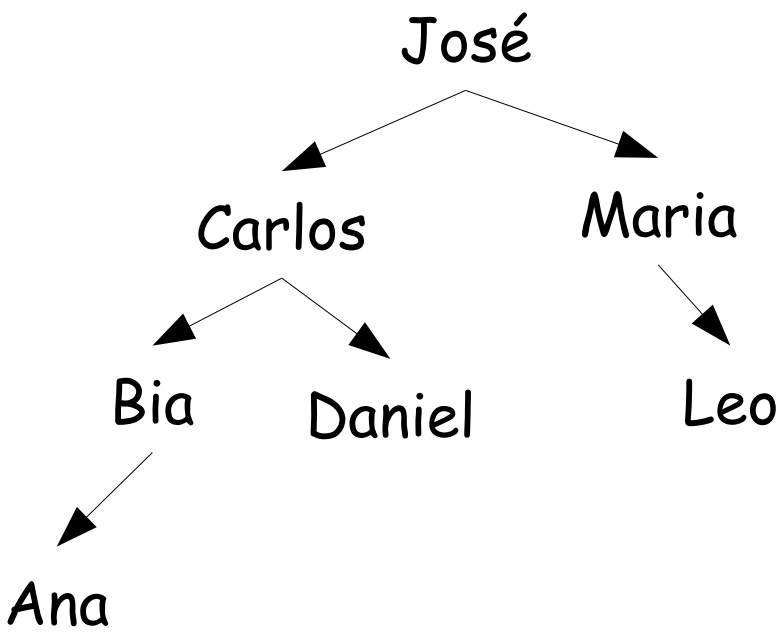
Acessos:

2

Contando acessos

Chamamos Maria

Verificando nossa escolha



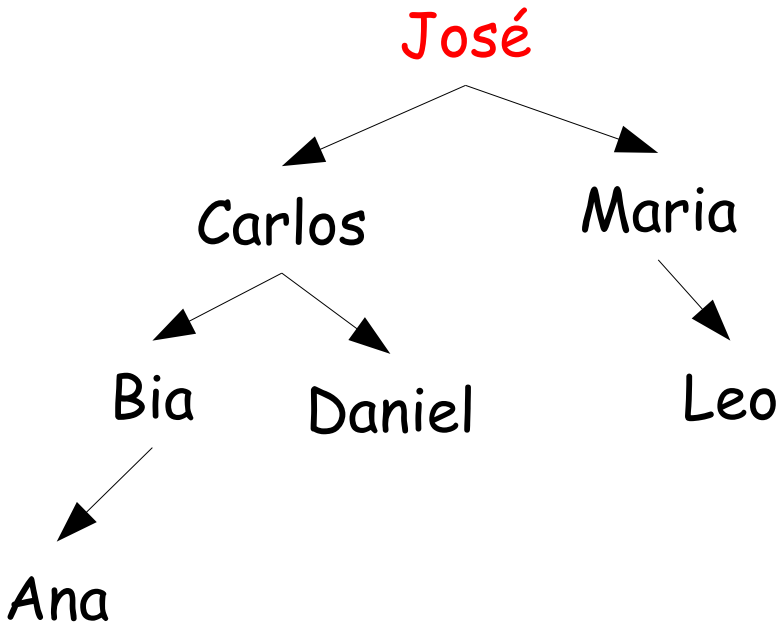
Acessos:

2

Contando acessos

Chamamos Maria

Verificando nossa escolha



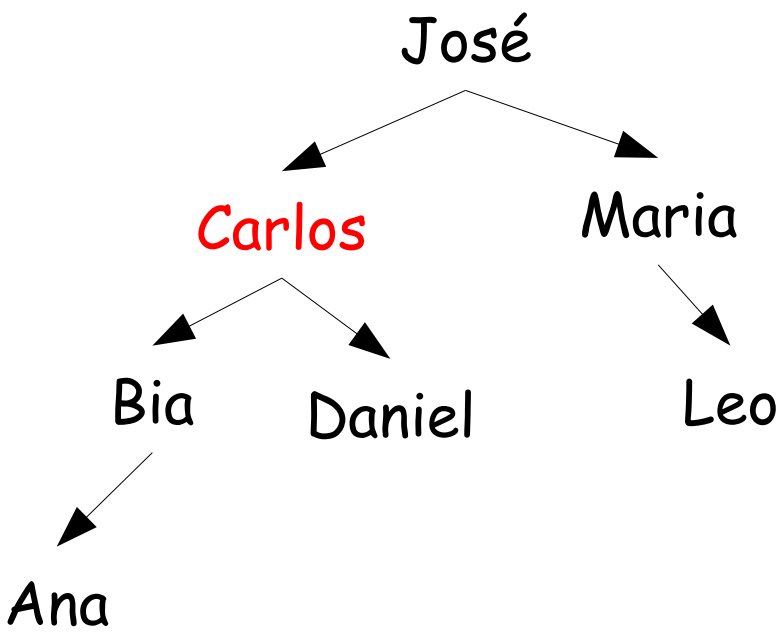
Acessos:

3

Contando acessos

Chamamos Maria, Daniel

Verificando nossa escolha



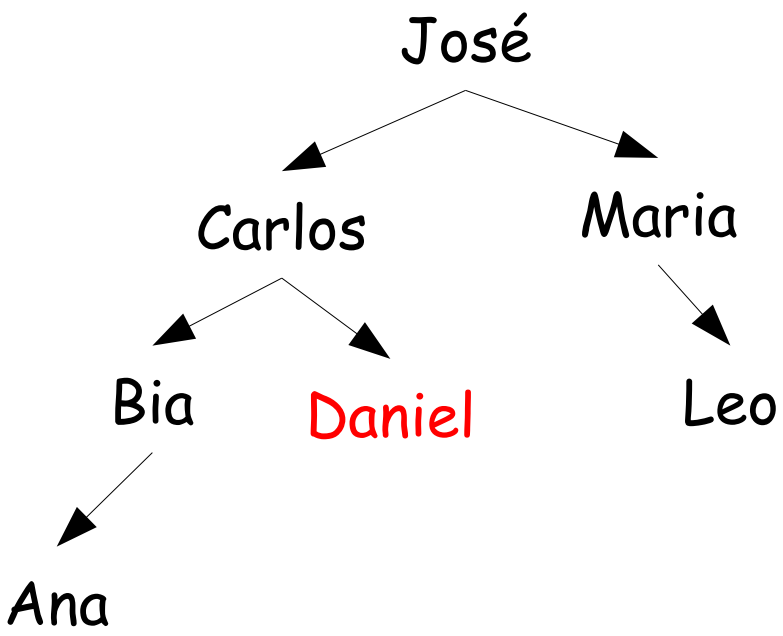
Acessos:

4

Contando acessos

Chamamos Maria, Daniel

Verificando nossa escolha



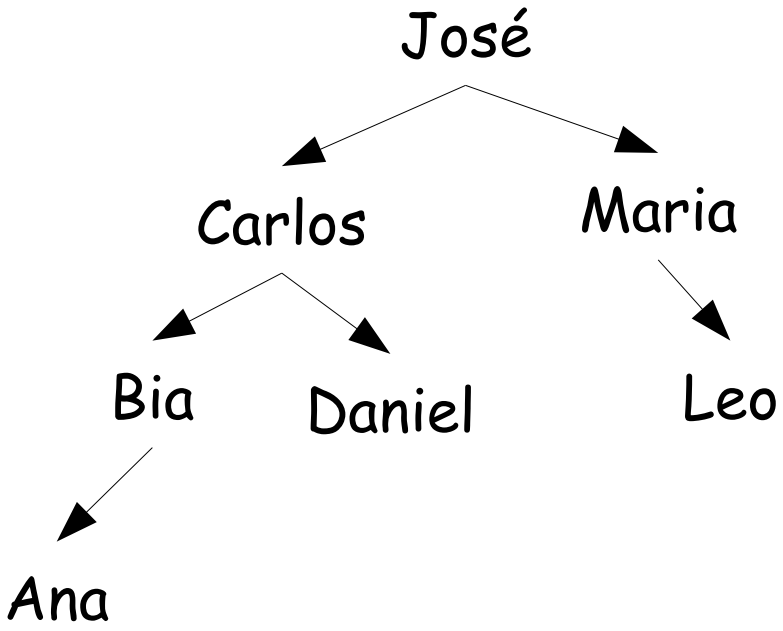
Acessos:

5

Contando acessos

Chamamos Maria, Daniel

Verificando nossa escolha



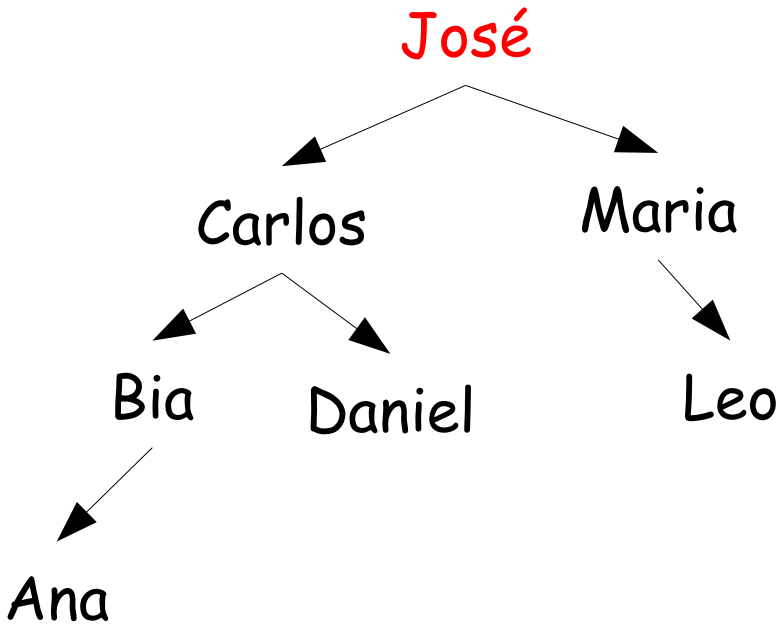
Acessos:

5

Contando acessos

Chamamos Maria, Daniel

Verificando nossa escolha



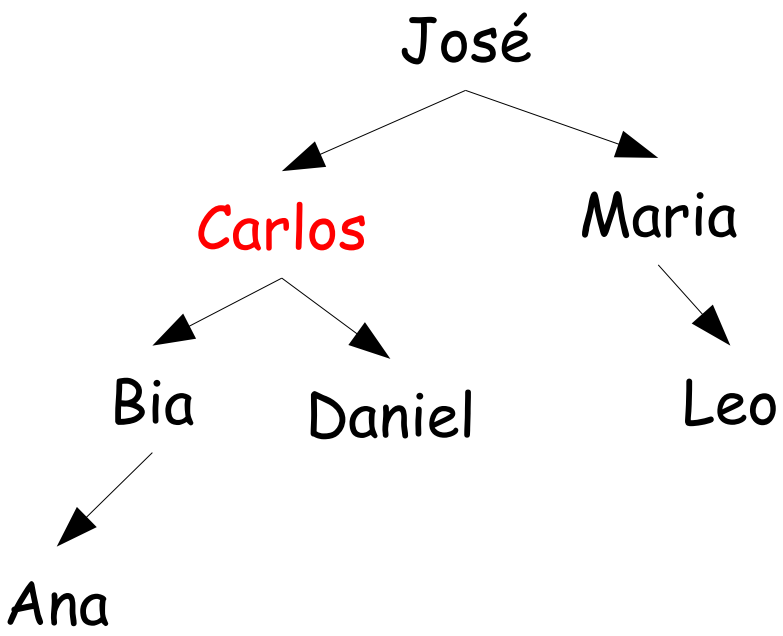
Acessos:

6

Contando acessos

Chamamos Maria, Daniel, Ana

Verificando nossa escolha



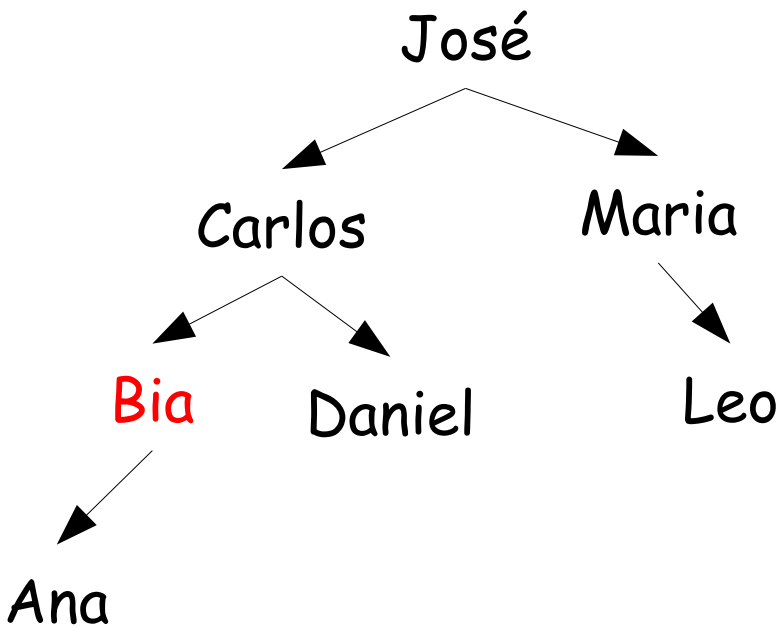
Acessos:

7

Contando acessos

Chamamos Maria, Daniel, Ana

Verificando nossa escolha



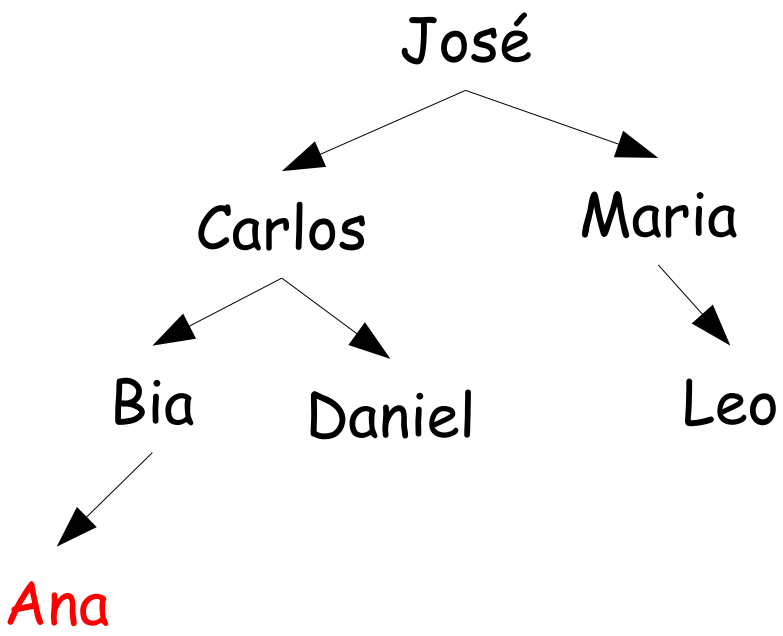
Acessos:

8

Contando acessos

Chamamos Maria, Daniel, Ana

Verificando nossa escolha



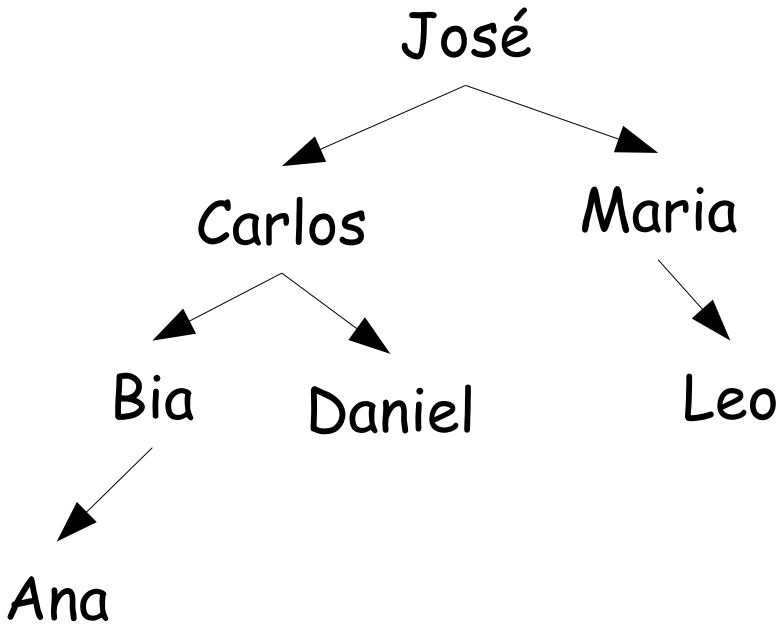
Acessos:

9

Contando acessos

Chamamos Maria, Daniel, Ana

Verificando nossa escolha



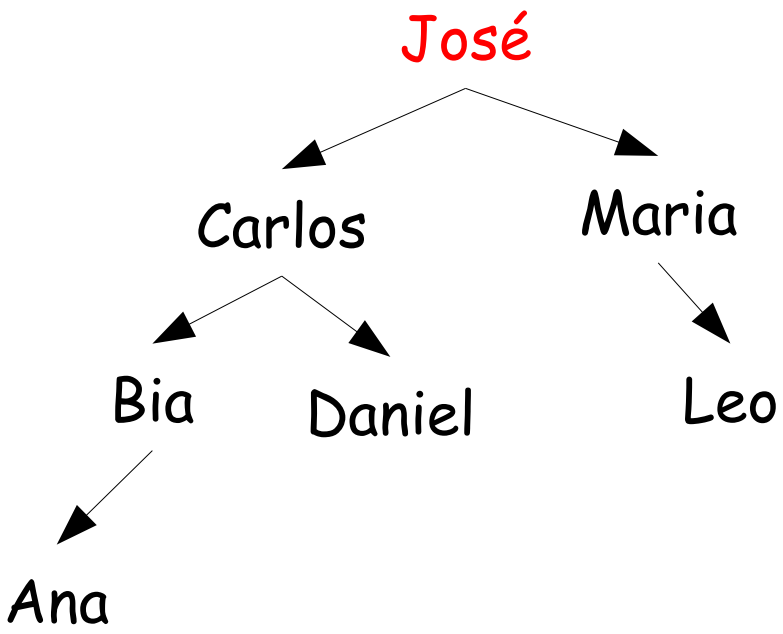
Acessos:

9

Contando acessos

Chamamos Maria, Daniel, Ana

Verificando nossa escolha



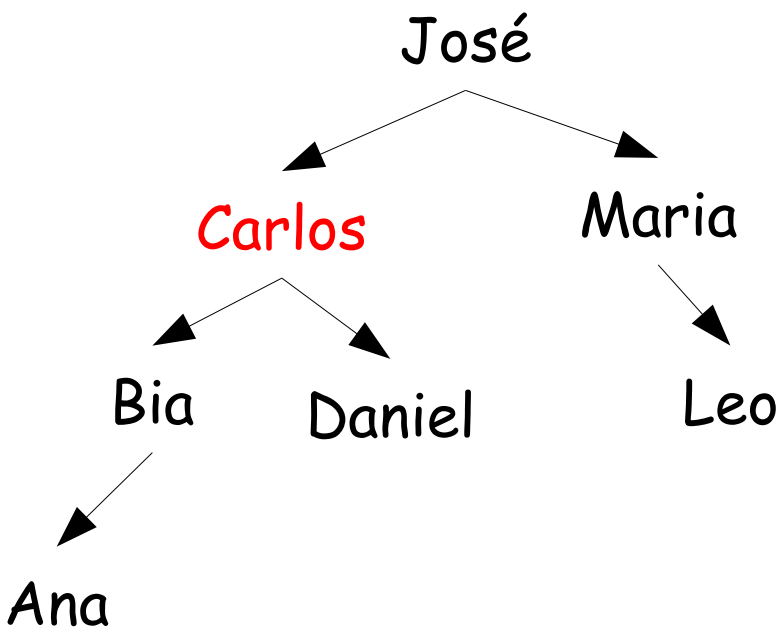
Acessos:

10

Contando acessos

Chamamos Maria, Daniel, Ana, Ana

Verificando nossa escolha



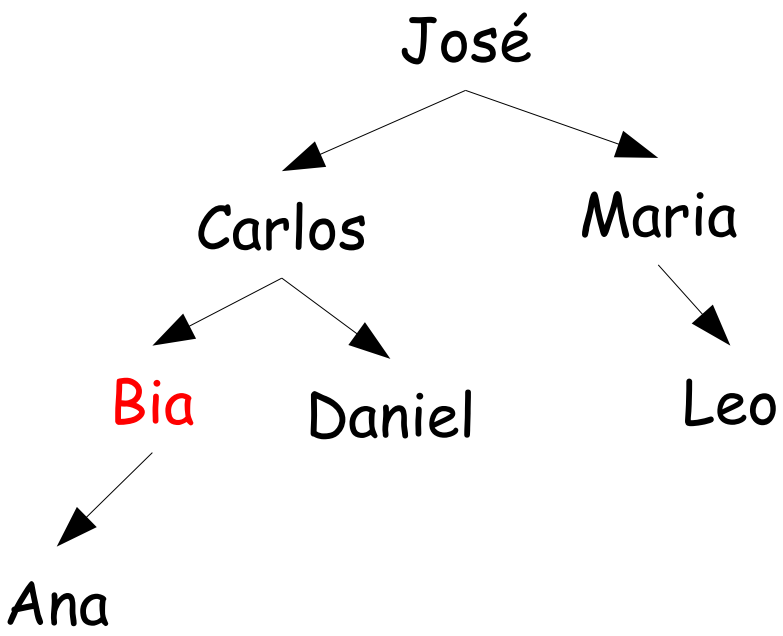
Acessos:

11

Contando acessos

Chamamos Maria, Daniel, Ana, Ana

Verificando nossa escolha



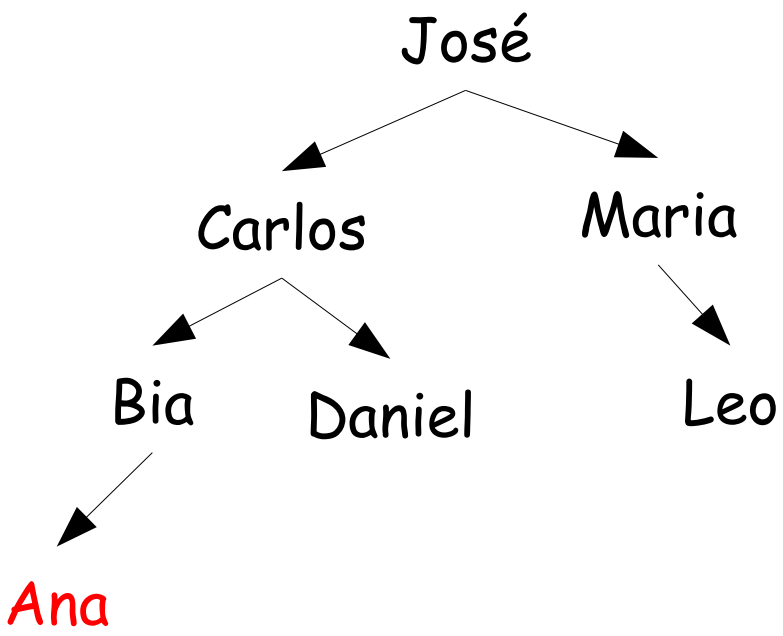
Acessos:

12

Contando acessos

Chamamos Maria, Daniel, Ana, Ana

Verificando nossa escolha



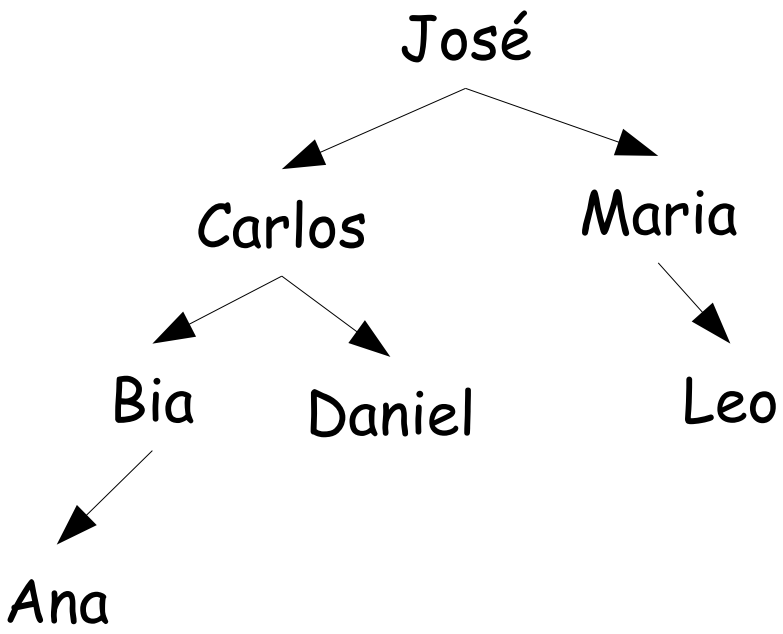
Acessos:

13

Contando acessos

Chamamos Maria, Daniel, Ana, Ana

Verificando nossa escolha



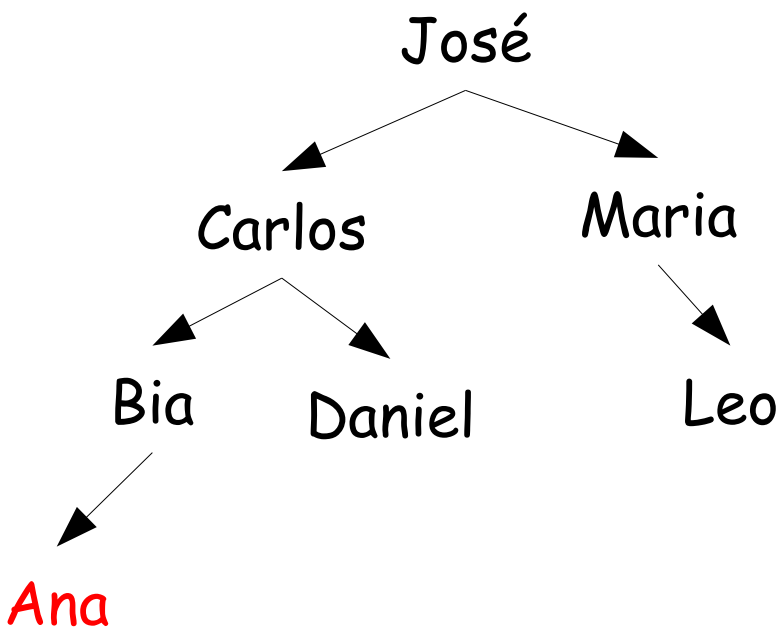
Acessos:

13

Contando acessos

Chamamos Maria, Daniel, Ana, Ana

Verificando nossa escolha



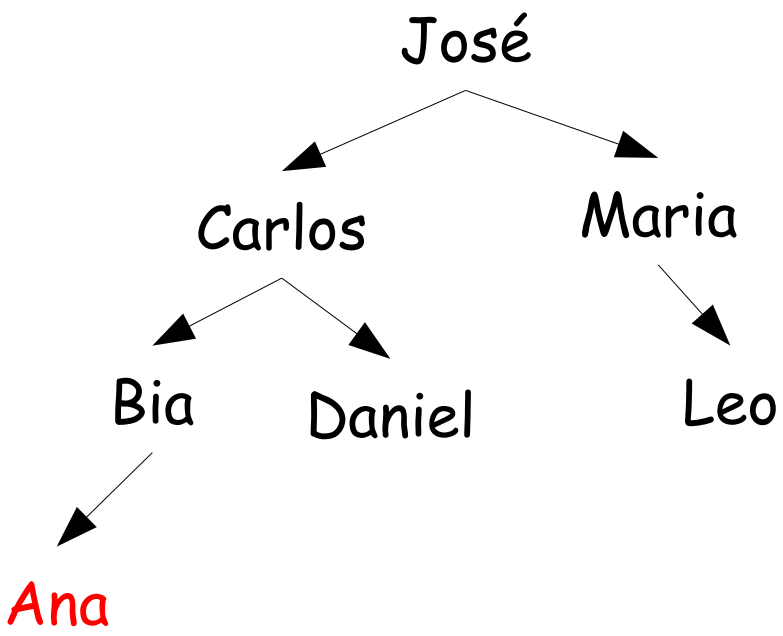
Acessos:

17

Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana

Verificando nossa escolha



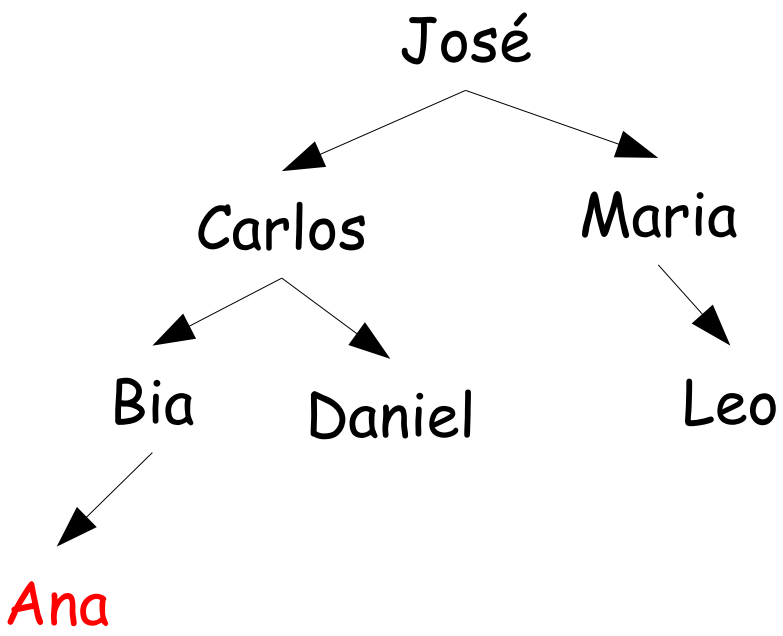
Acessos:

21

Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana

Verificando nossa escolha



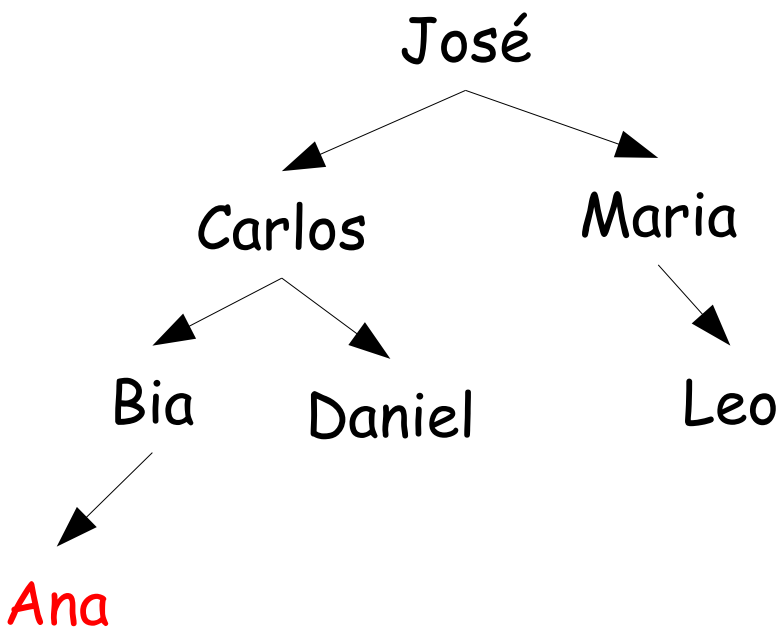
Acessos:

25

Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana

Verificando nossa escolha



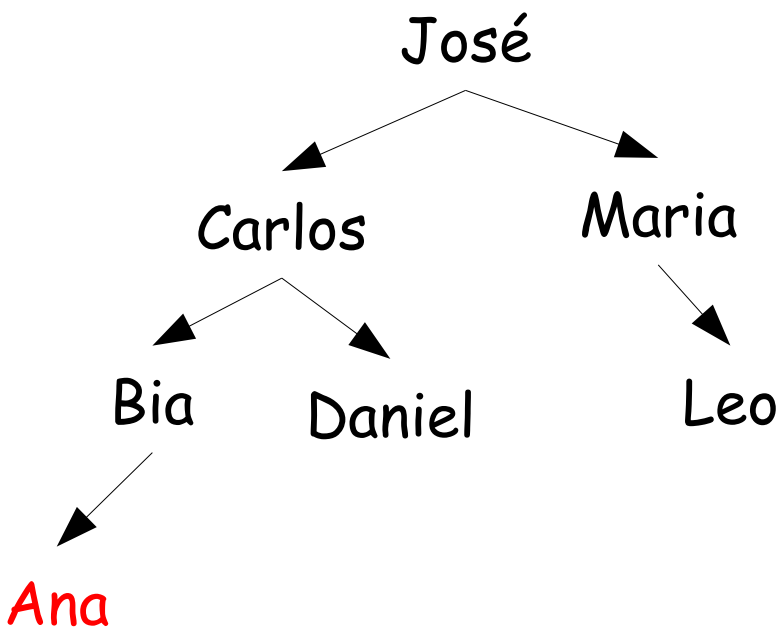
Acessos:

33

Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana e Ana

Verificando nossa escolha



Acessos:

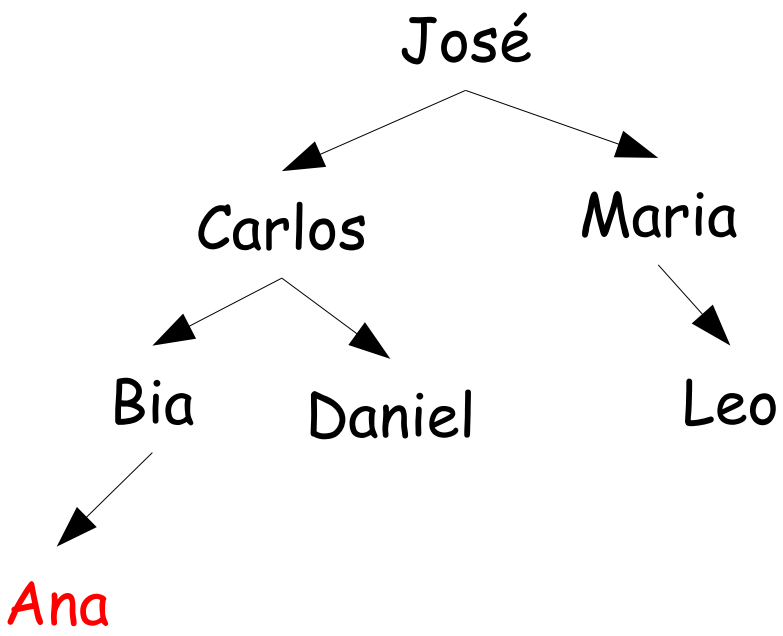
33

Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana e Ana

Diagnóstico:

Verificando nossa escolha



Acessos:

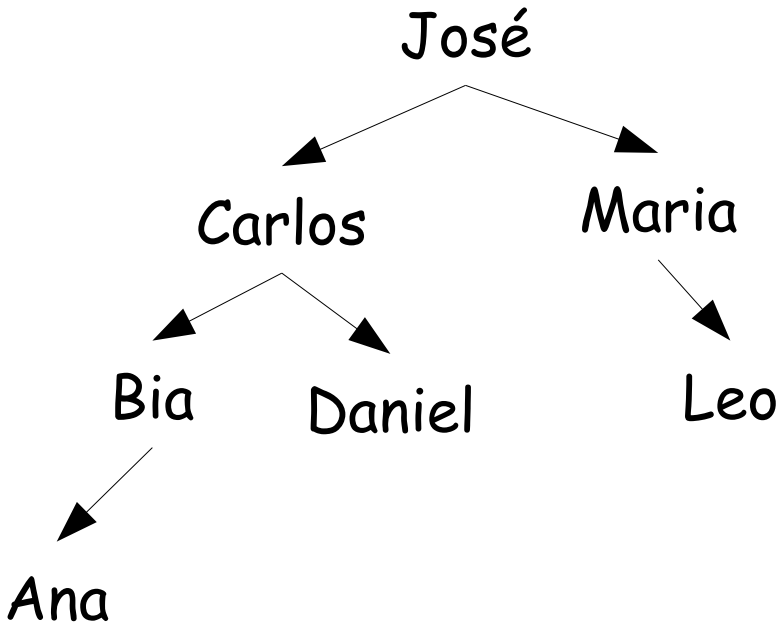
33

Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana e Ana

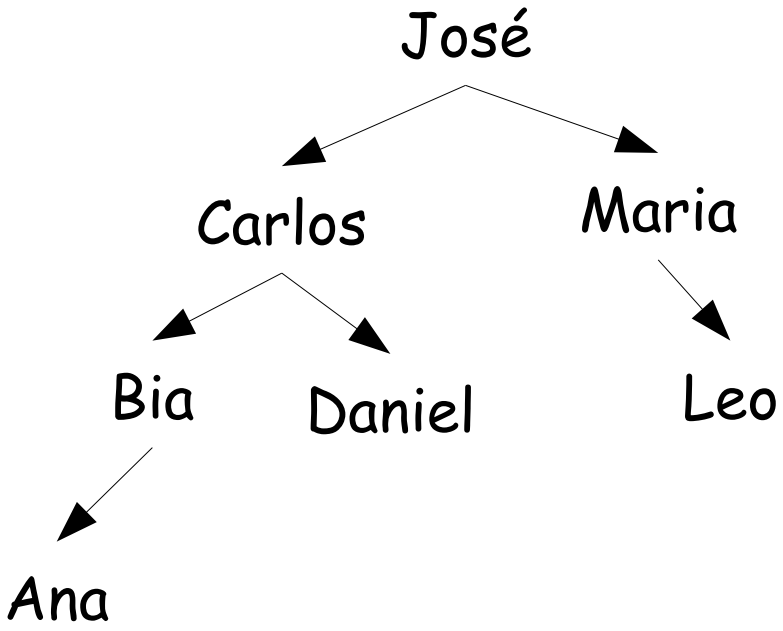
Diagnóstico: Acessamos um certo elemento várias vezes!

Modificando a árvore



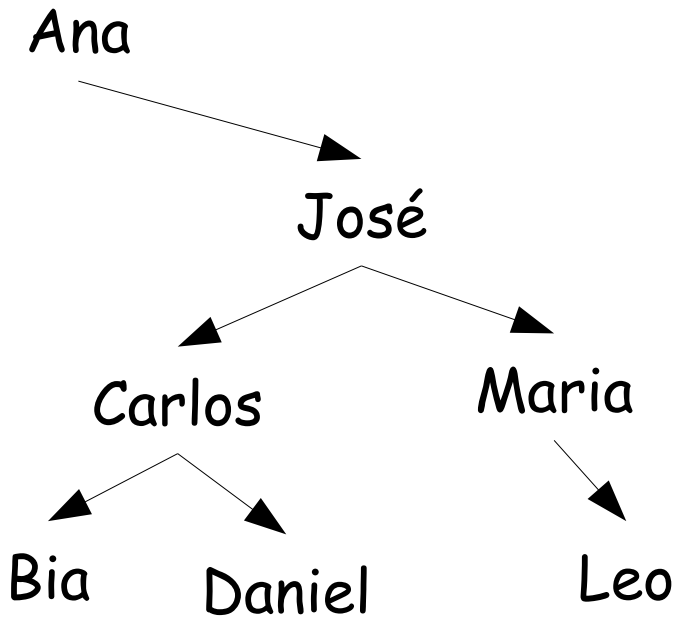
Vamos modificar a árvore:

Modificando a árvore



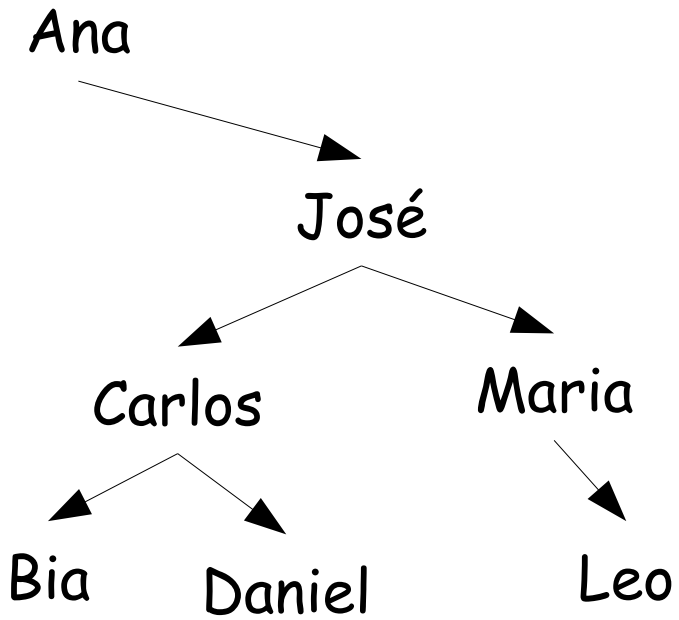
Vamos modificar a árvore: trazemos o elemento mais recente para cima!

Modificando a árvore



Vamos modificar a árvore: trazemos o elemento mais recente para cima!

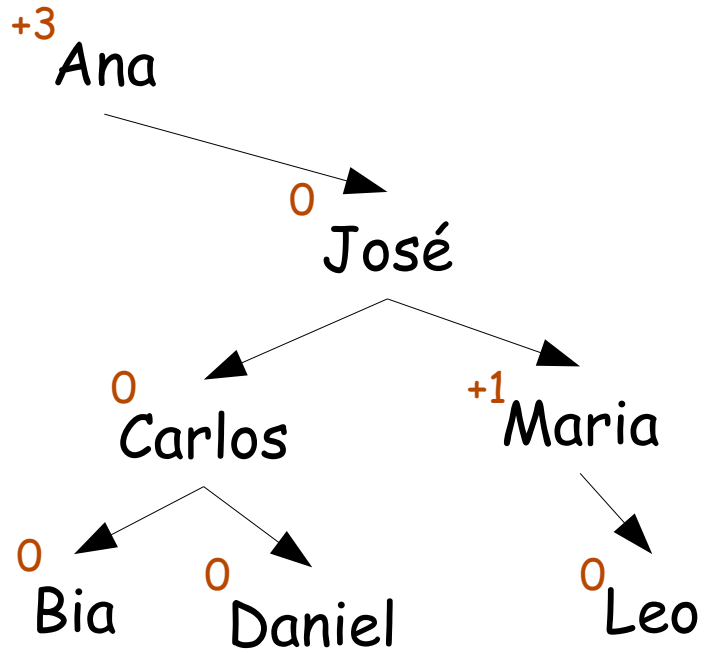
Modificando a árvore



Vamos modificar a árvore: trazemos o elemento mais recente para cima!

Pergunta: A árvore é balanceada?

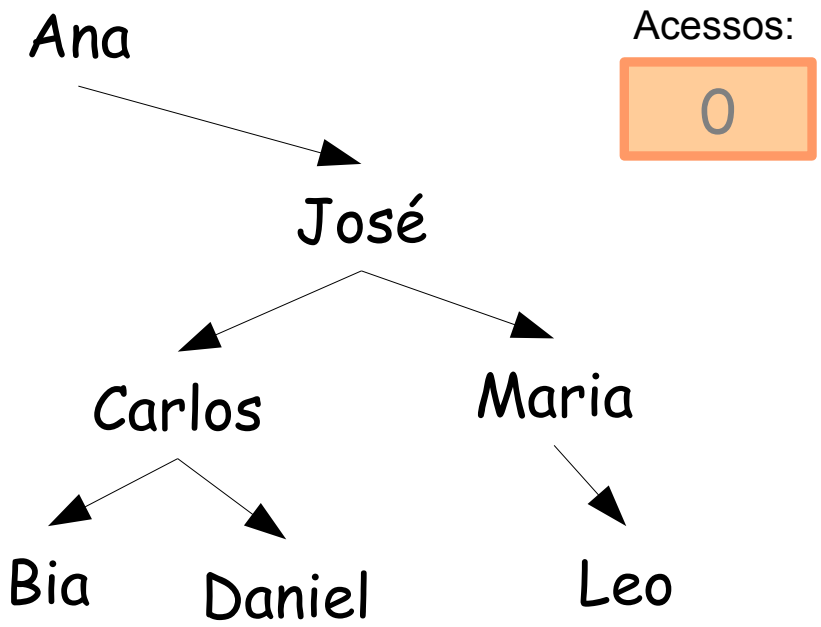
Modificando a árvore



Vamos modificar a árvore: trazemos o elemento mais recente para cima!

Pergunta: A árvore é balanceada? Não!

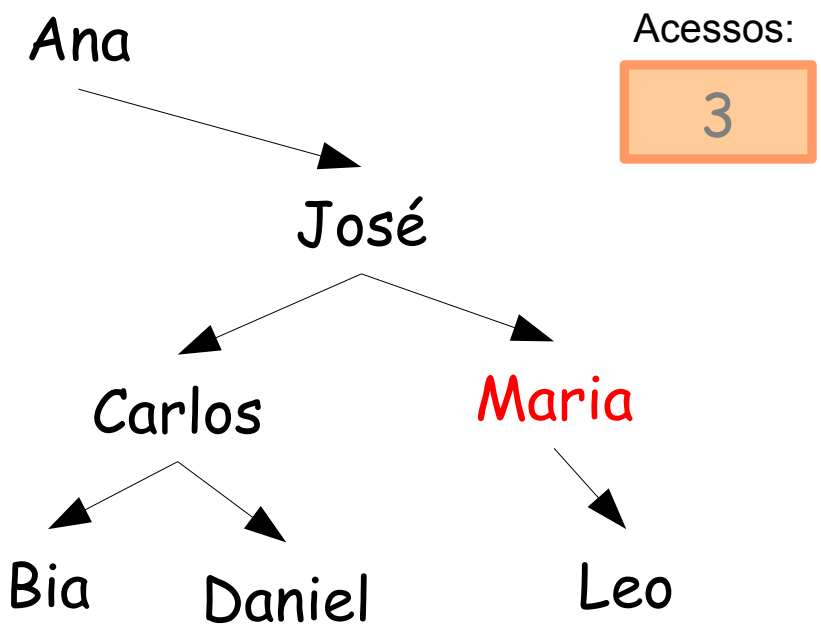
Contando de novo



Contando acessos

Chamamos

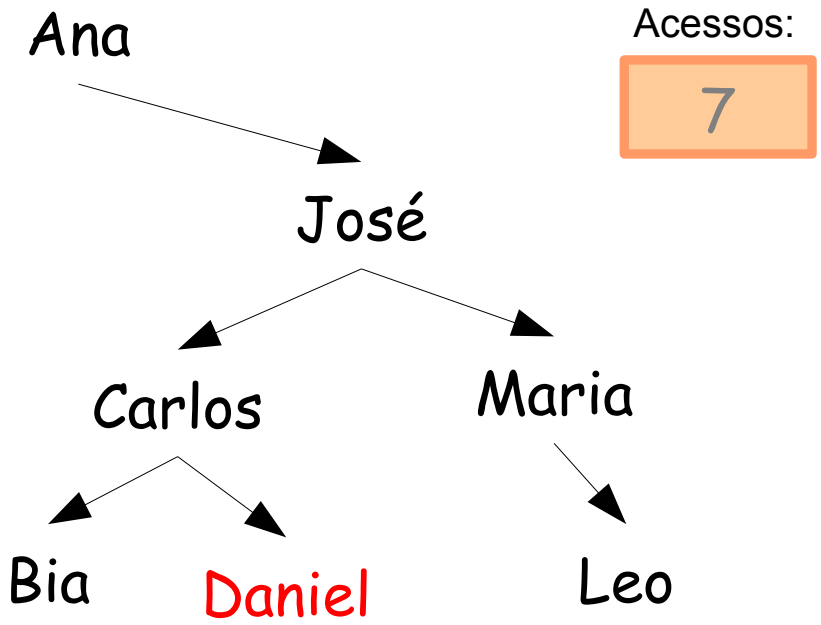
Contando de novo



Contando acessos

Chamamos Maria

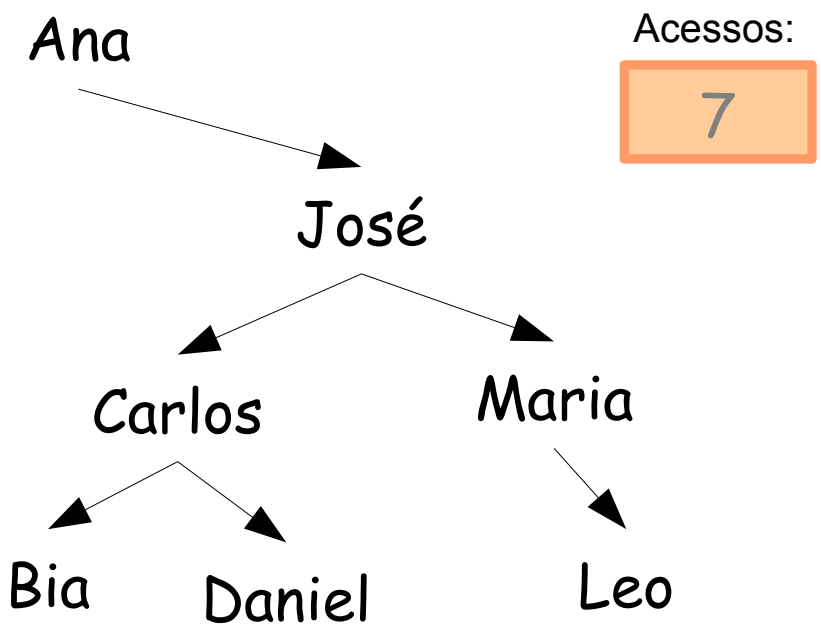
Contando de novo



Contando acessos

Chamamos Maria, Daniel

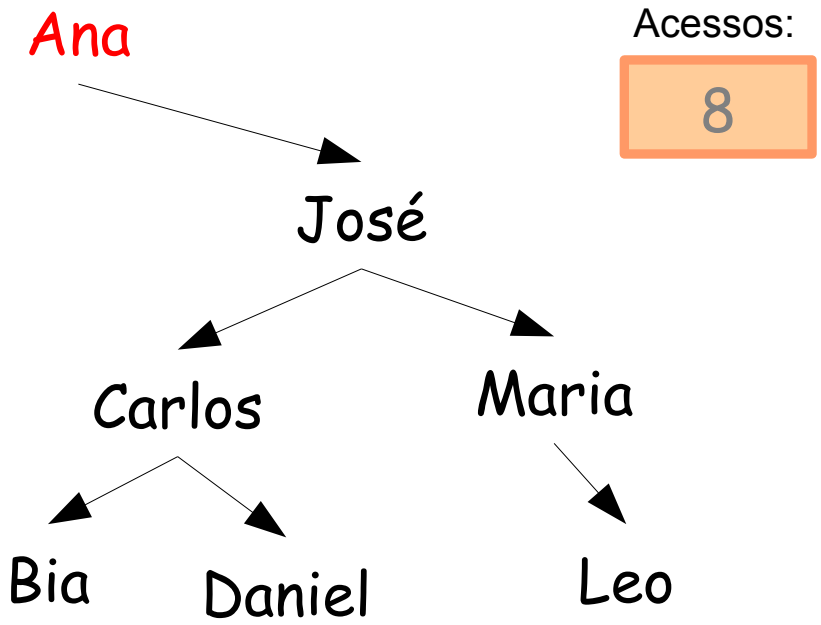
Contando de novo



Contando acessos

Chamamos Maria, Daniel

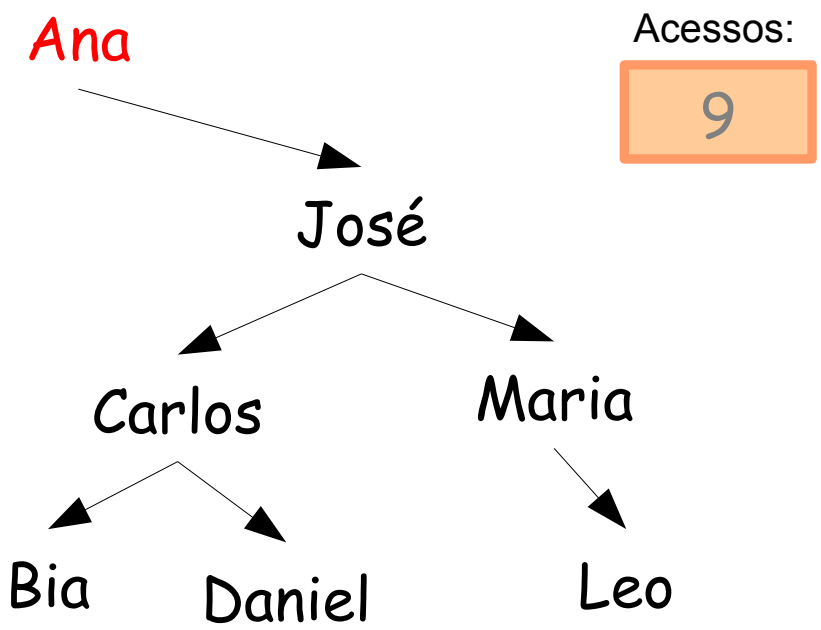
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana

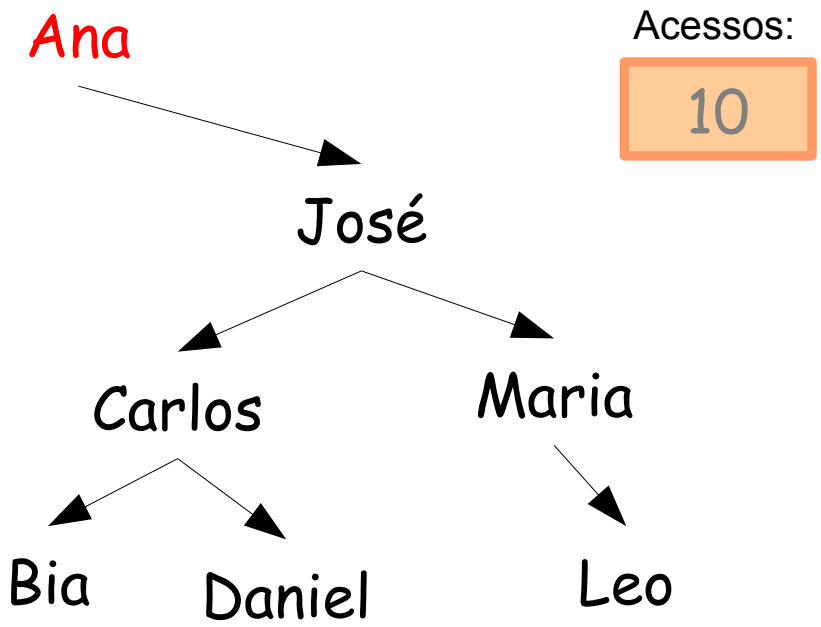
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana

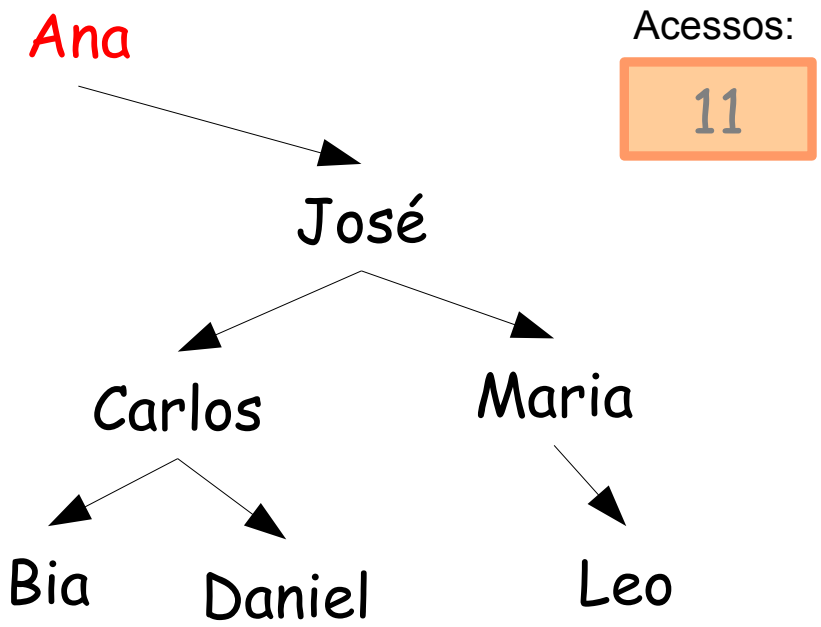
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana

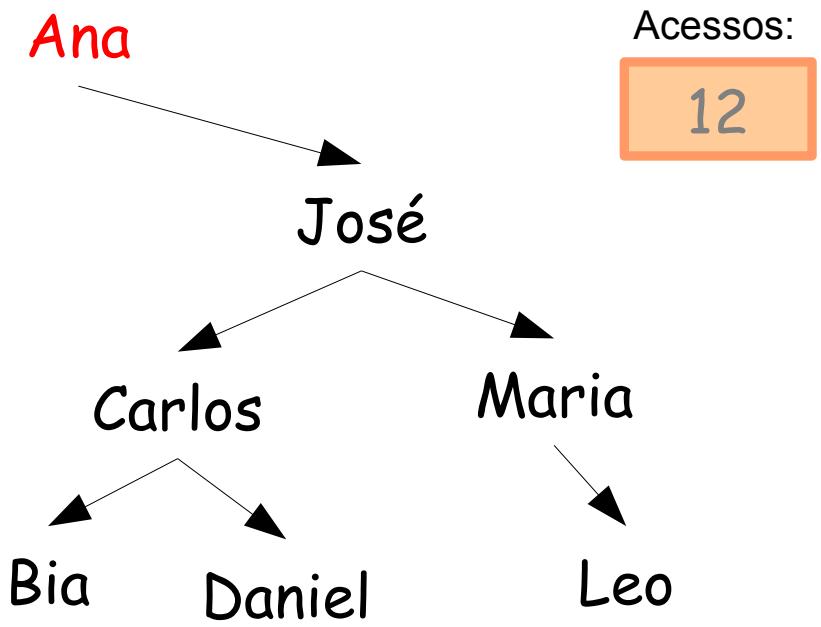
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana

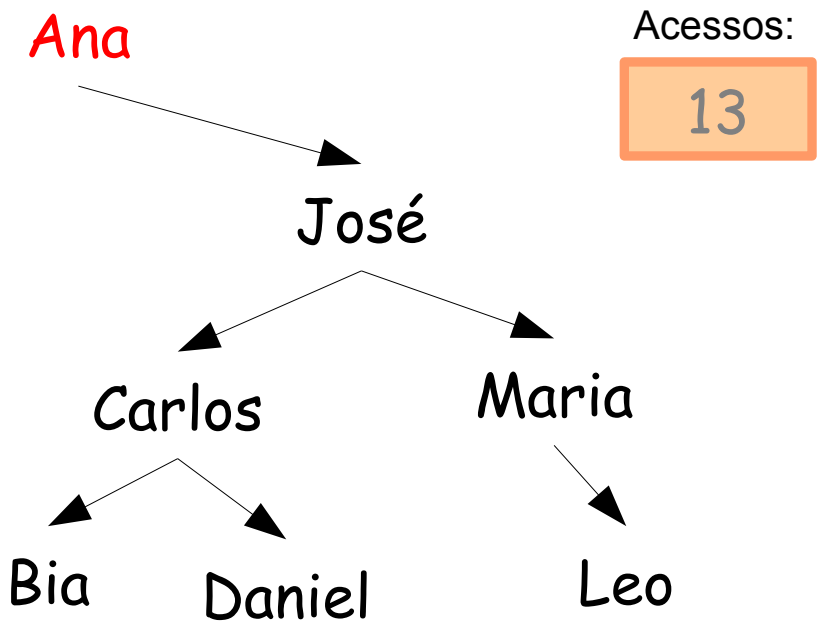
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana

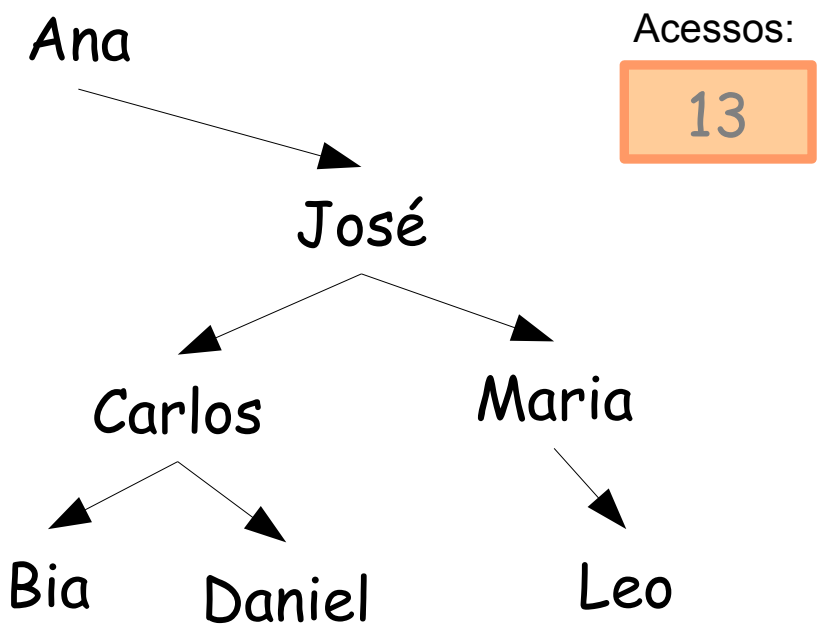
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana

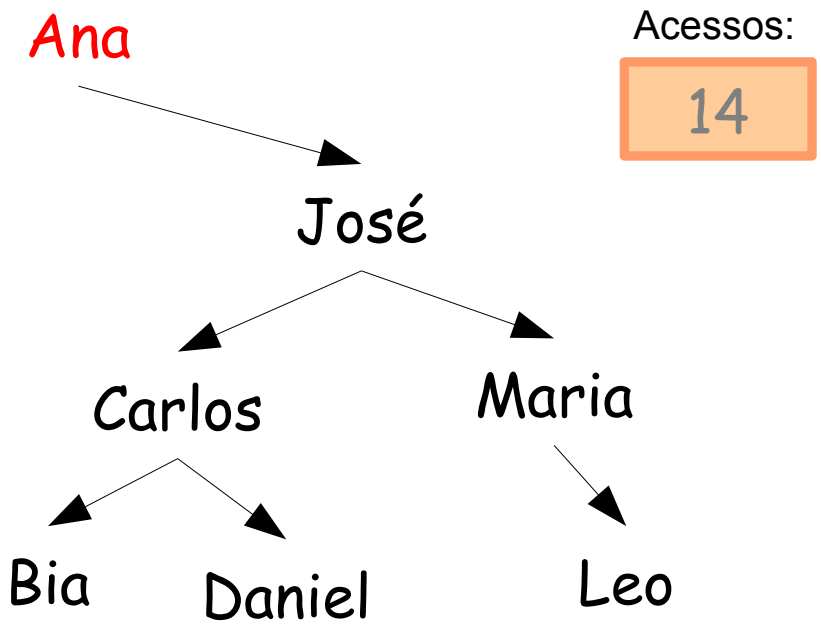
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana

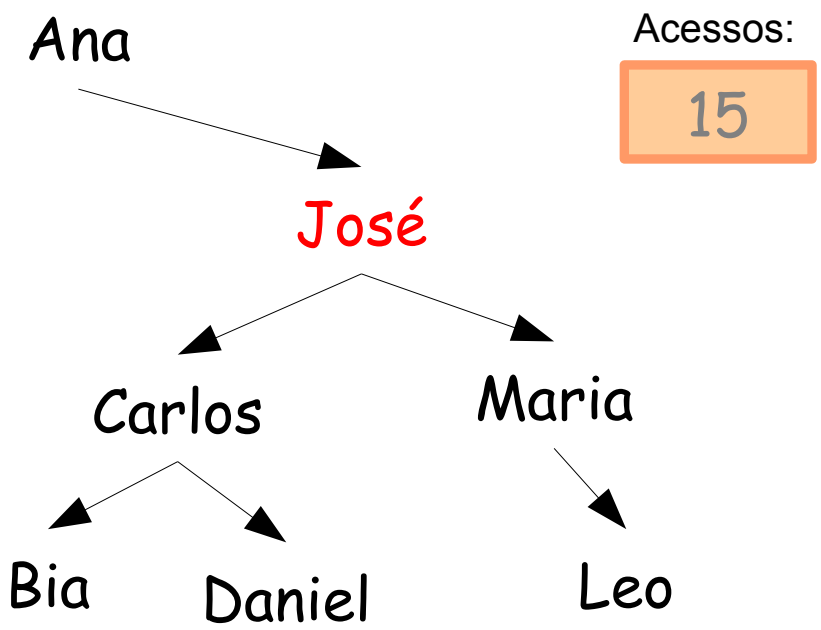
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria

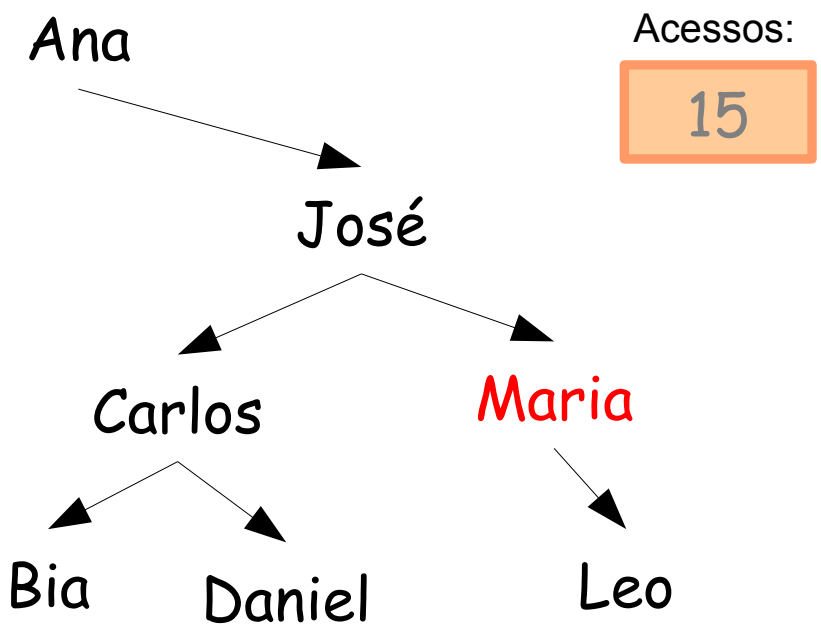
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria

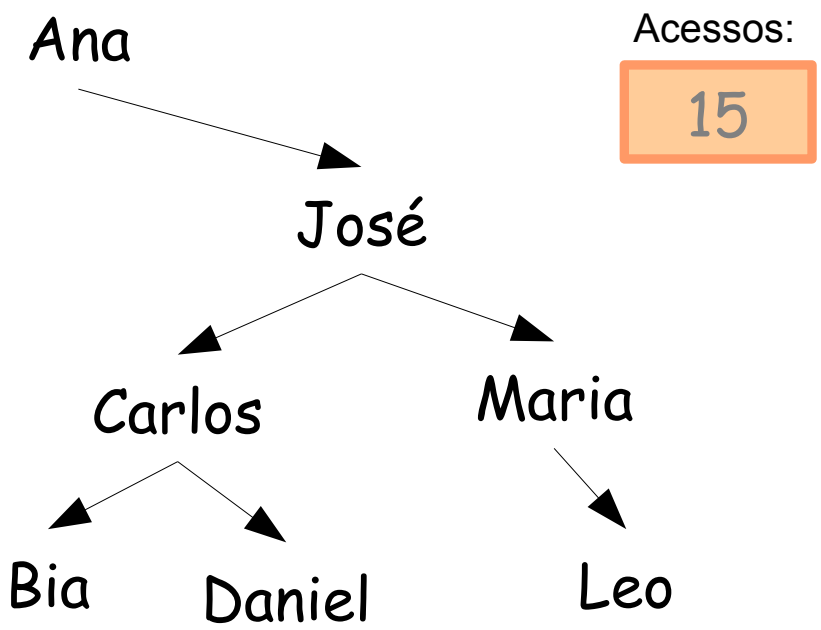
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria

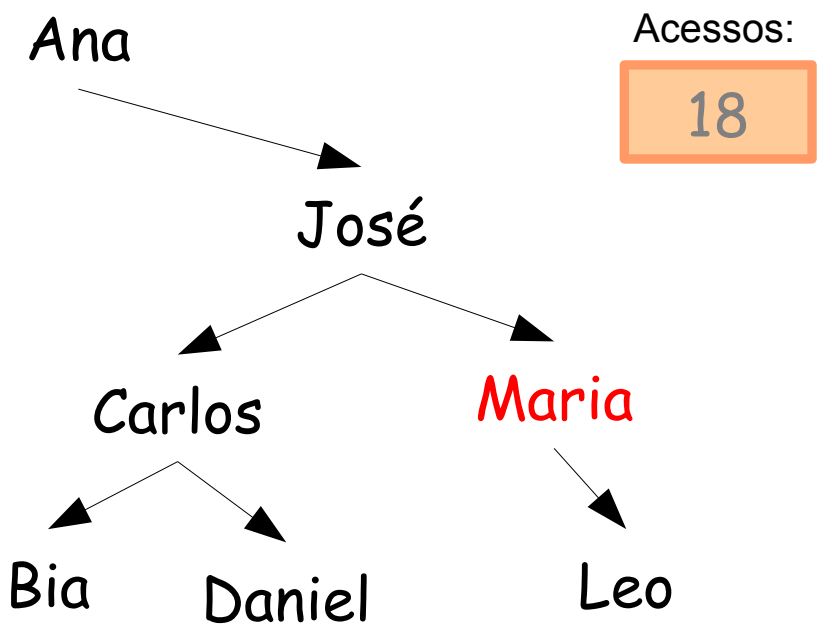
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria

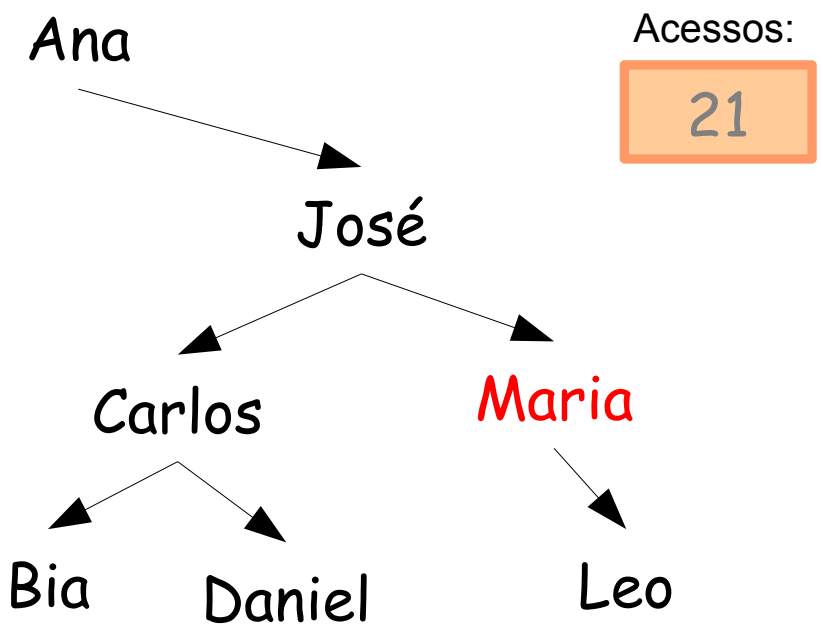
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria

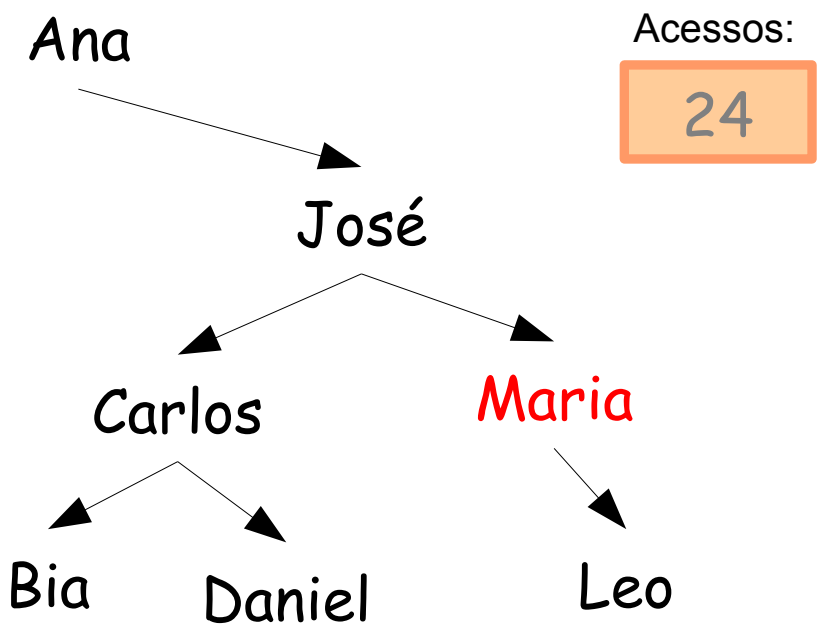
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria

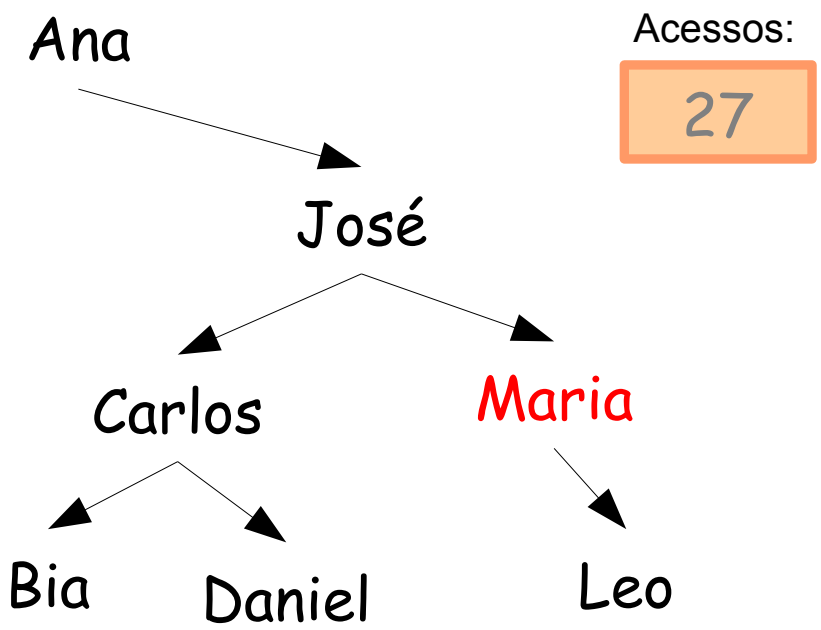
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria, Maria

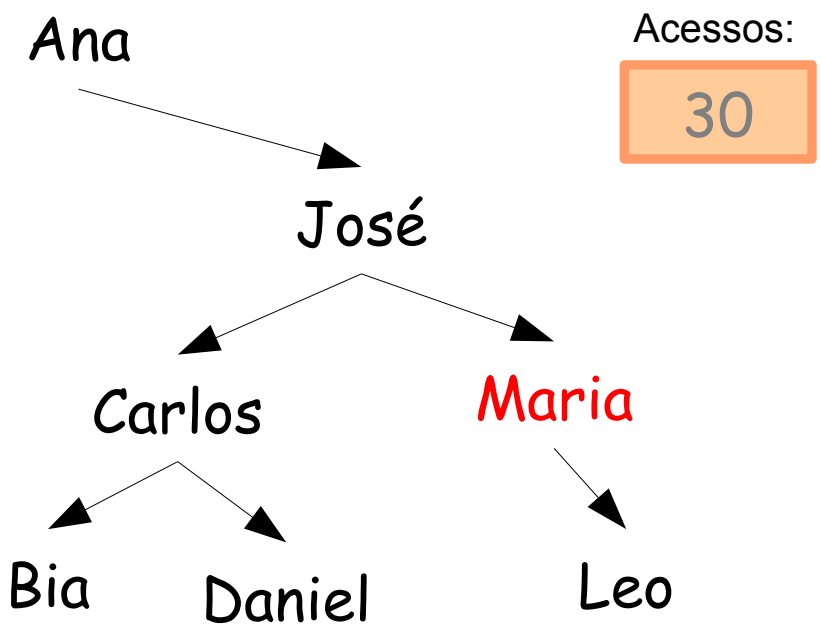
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria, Maria, Maria...

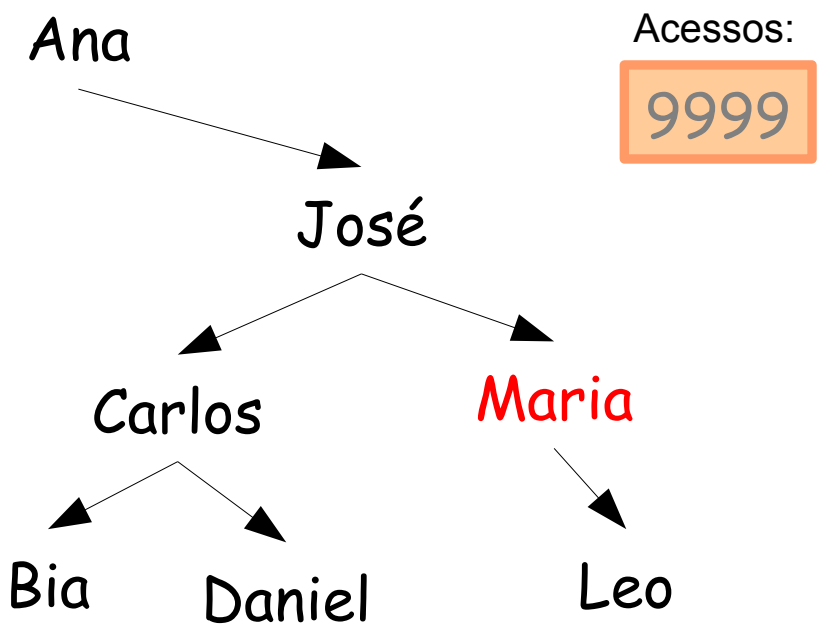
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria, Maria, Maria...

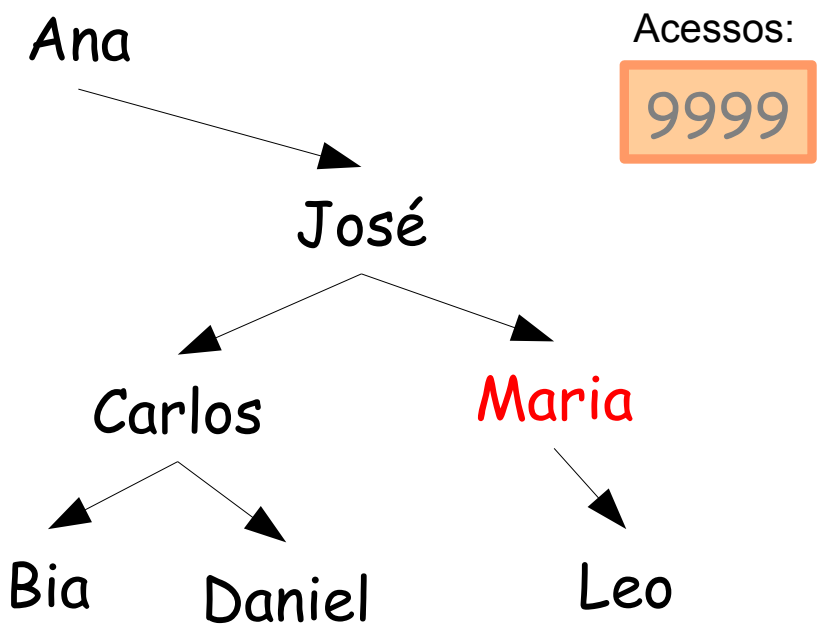
Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria, Maria, Maria...

Contando de novo

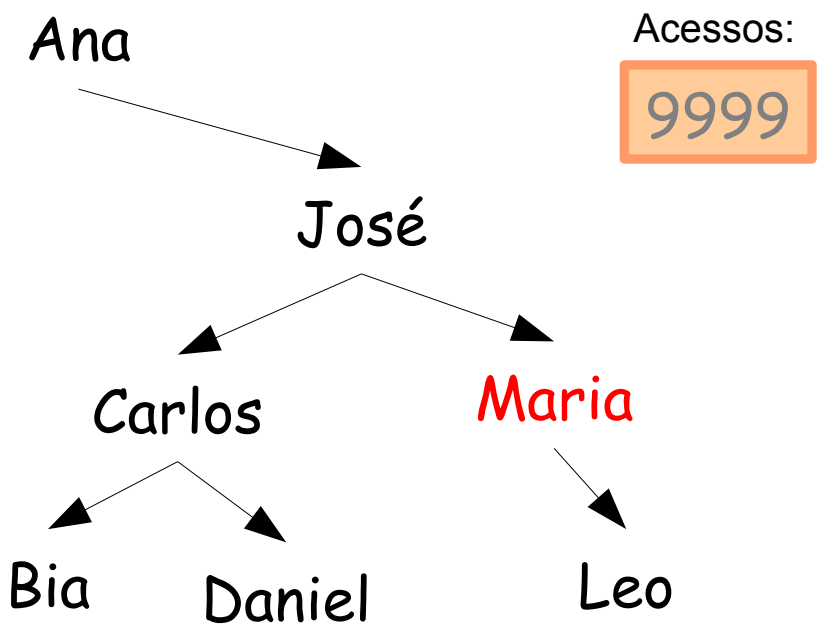


Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria, Maria, Maria...

Diagnóstico:

Contando de novo



Contando acessos

Chamamos Maria, Daniel, Ana, Ana, Ana, Ana, Ana, Ana, Maria, Maria, Maria, Maria, Maria...

Diagnóstico: Acessamos os mais recentes várias vezes!

Localidade de Referência

Princípio da localidade

É o princípio que um mesmo elemento que um certo elemento é acessado muitas vezes por um curto período de tempo. Acontece de várias maneiras:

Localidade de Referência

Princípio da localidade

É o princípio que um mesmo elemento que um certo elemento é acessado muitas vezes por um curto período de tempo. Acontece de várias maneiras:

- **temporal:** um elemento acessado recentemente tende a ser acessado novamente em breve
- **espacial:** se um dado foi acessado em certa parte da memória, locais próximos tendem a ser acessados em breve

Localidade de Referência

Princípio da localidade

É o princípio que um mesmo elemento que um certo elemento é acessado muitas vezes por um curto período de tempo. Acontece de várias maneiras:

- **temporal:** um elemento acessado recentemente tende a ser acessado novamente em breve
- **espacial:** se um dado foi acessado em certa parte da memória, locais próximos tendem a ser acessados em breve

Exemplos

- conjuntos dinâmicos
- código de um programa
- dados de um registro

Árvore de Afunilamento (*Splay Tree*)

Splay Tree

- introduzida por Daniel Dominic Sleator and Robert Endre Tarjan (1985)
- tenta fazer com que elementos mais recentemente acessados estejam perto da raiz

Árvore de Afunilamento (*Splay Tree*)

Splay Tree

- introduzida por Daniel Dominic Sleator and Robert Endre Tarjan (1985)
- tenta fazer com que elementos mais recentemente acessados estejam perto da raiz

Definição

Um Splay Tree é alguma árvore de busca cuja implementação usa a operação de afunilamento.

Árvore de Afunilamento (*Splay Tree*)

Splay Tree

- introduzida por Daniel Dominic Sleator and Robert Endre Tarjan (1985)
- tenta fazer com que elementos mais recentemente acessados estejam perto da raiz

Definição

Um Splay Tree é alguma árvore de busca cuja implementação usa a operação de afunilamento.

Característica

- **Pior caso de uma busca:** $O(n)$
- **Pior caso de k operações consecutivas:** $O(k \log(n))$

Principal operação: **afunilar**

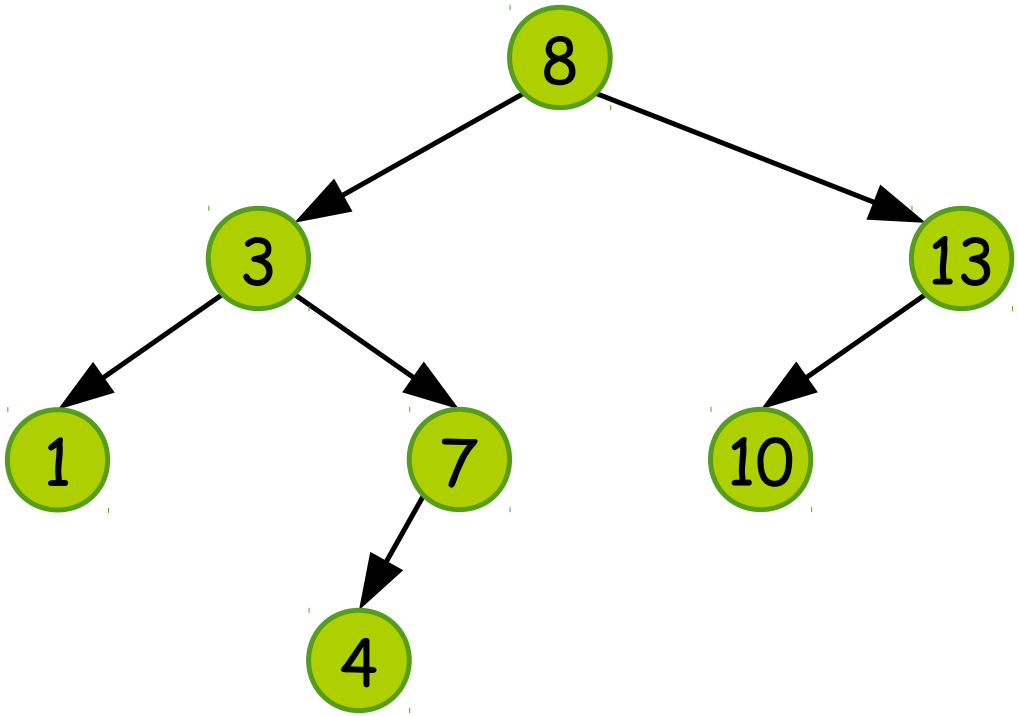
Afunilar

Afunilar um nó é um operação que transforma um né em raiz.

Principal operação: **afunilar**

Afunilar

Afunilar um nó é um operação que transforma um né em raiz.

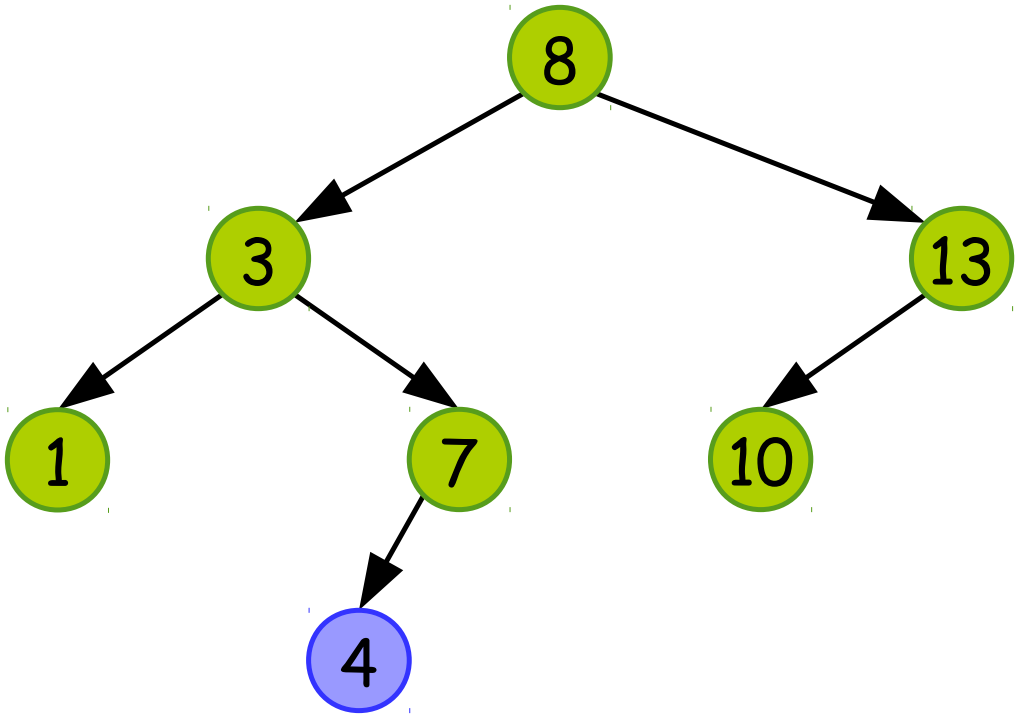


Exemplo: Afunilar o 4.

Principal operação: **afunilar**

Afunilar

Afunilar um nó é um operação que transforma um né em raiz.

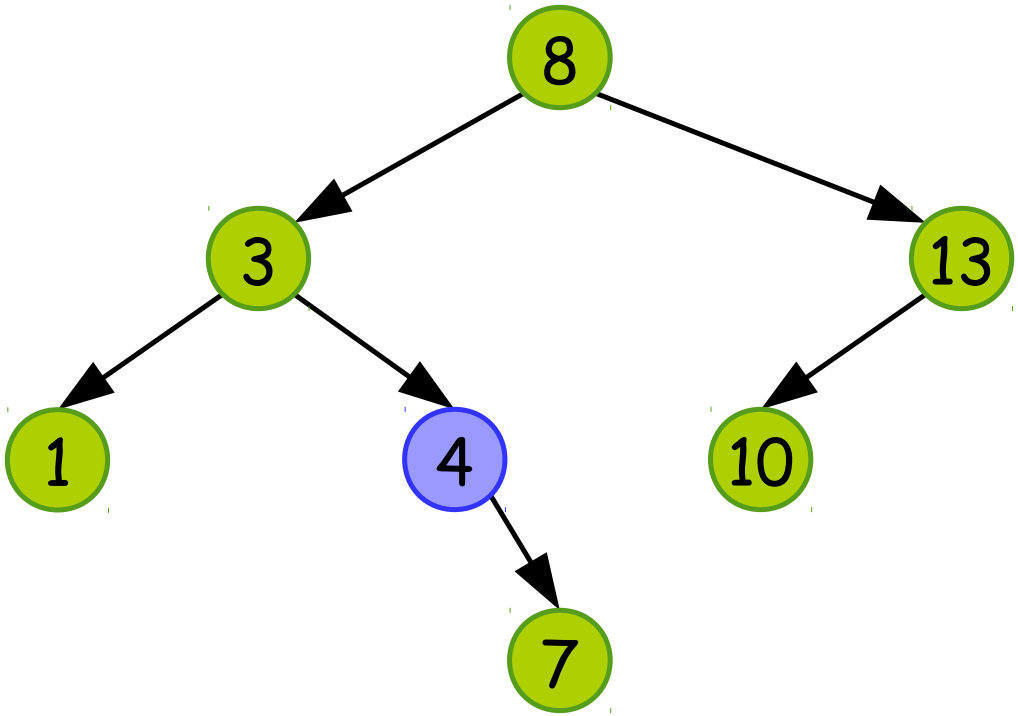


Exemplo: Afunilar o 4.

Principal operação: **afunilar**

Afunilar

Afunilar um nó é um operação que transforma um né em raiz.

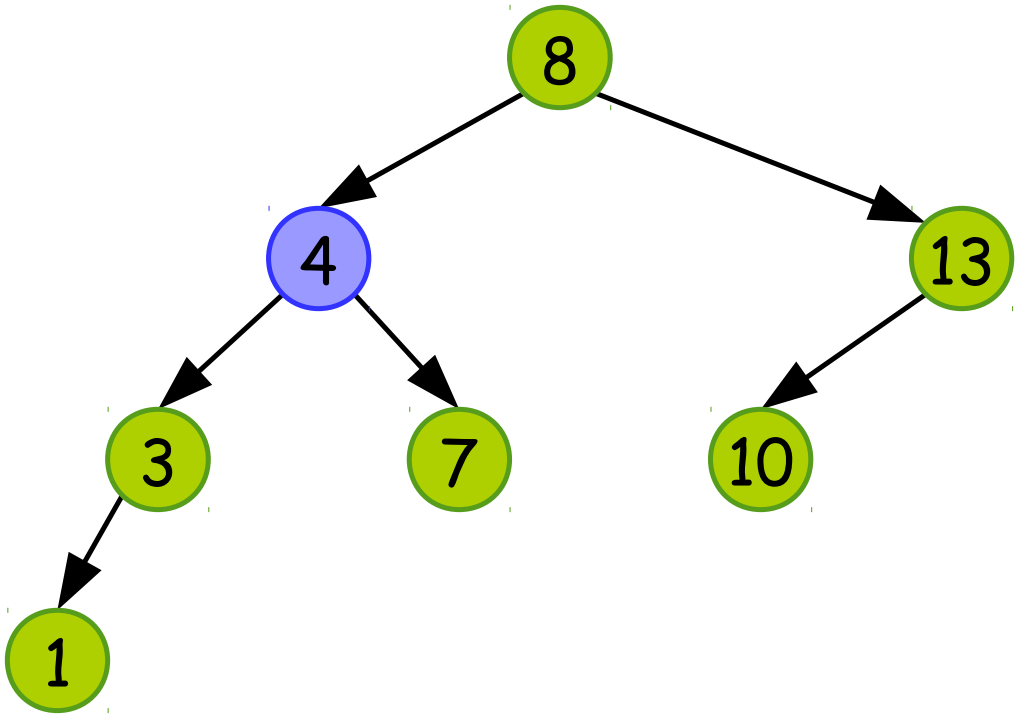


Exemplo: Afunilar o 4.

Principal operação: **afunilar**

Afunilar

Afunilar um nó é um operação que transforma um né em raiz.

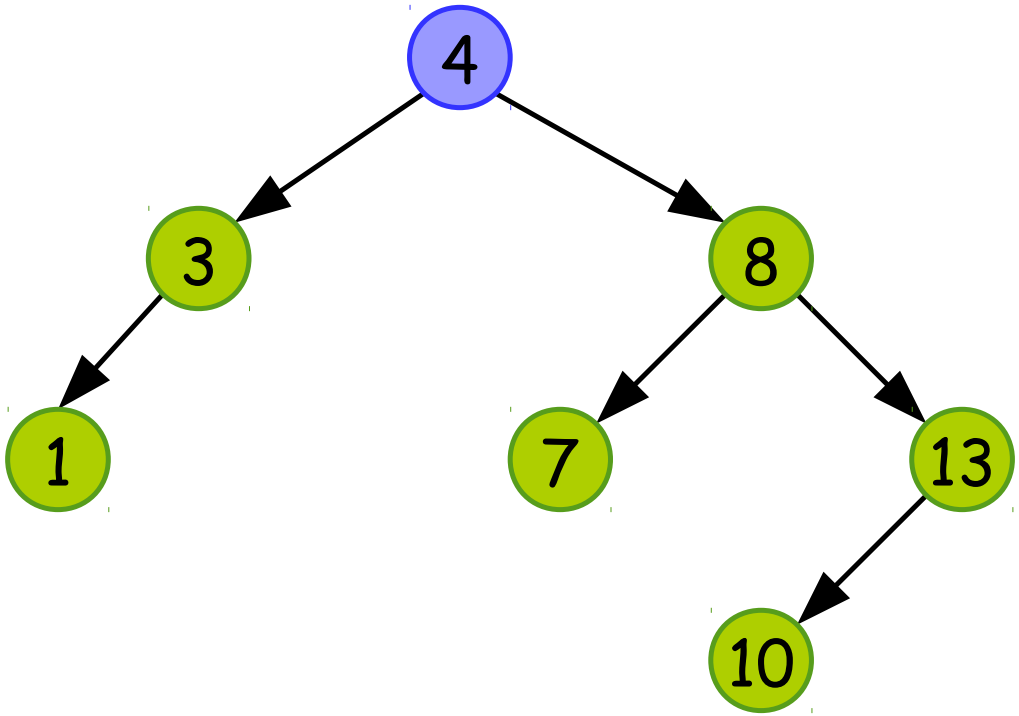


Exemplo: Afunilar o 4.

Principal operação: **afunilar**

Afunilar

Afunilar um nó é um operação que transforma um né em raiz.



Exemplo: Afunilar o 4.

Ideia

Afunilar nó x

Fazemos rotações dependendo do caso:

1. x é raiz
2. Caso *zig*: pai de x é raiz
3. Caso *zig-zig*: pai e avô alinhados
4. Caso *zig-zag*: pai e avô não alinhados

Ideia

Afunilar nó x

Fazemos rotações dependendo do caso:

1. x é raiz
2. Caso *zig*: pai de x é raiz
3. Caso *zig-zig*: pai e avô alinhados
4. Caso *zig-zag*: pai e avô não alinhados

Algoritmo: Enquanto x não for raiz, escolhe rotação adequada.

Ideia

Afunilar nó x

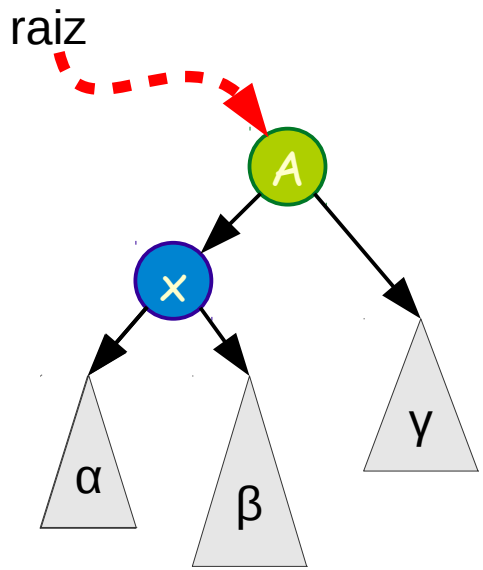
Fazemos rotações dependendo do caso:

1. x é raiz
2. Caso *zig*: pai de x é raiz
3. Caso *zig-zig*: pai e avô alinhados
4. Caso *zig-zag*: pai e avô não alinhados

Algoritmo: Enquanto x não for raiz, escolhe rotação adequada.

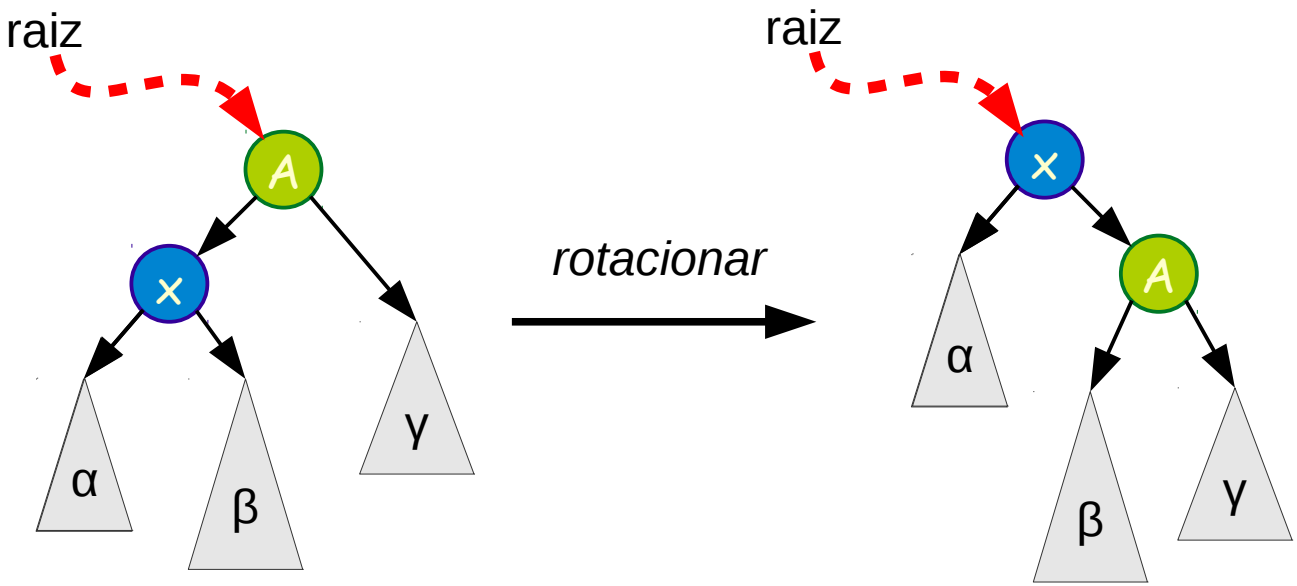
Observação: Nossa implementação usará nós com ponteiro para o nó pai!

Caso 2: *zig* (pai de x é raiz)



Procedimento

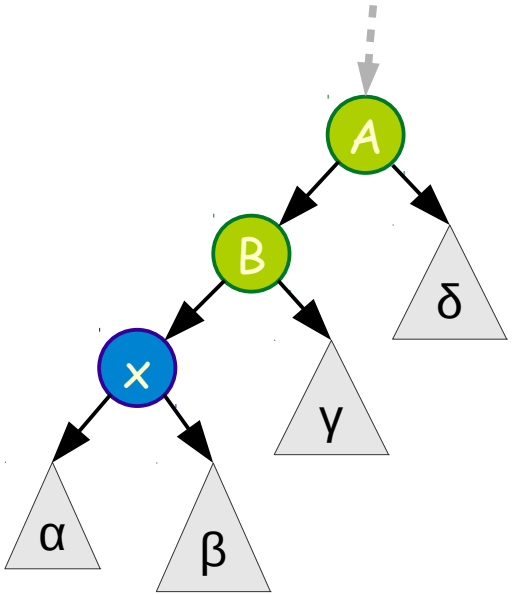
Caso 2: *zig* (pai de x é raiz)



Procedimento

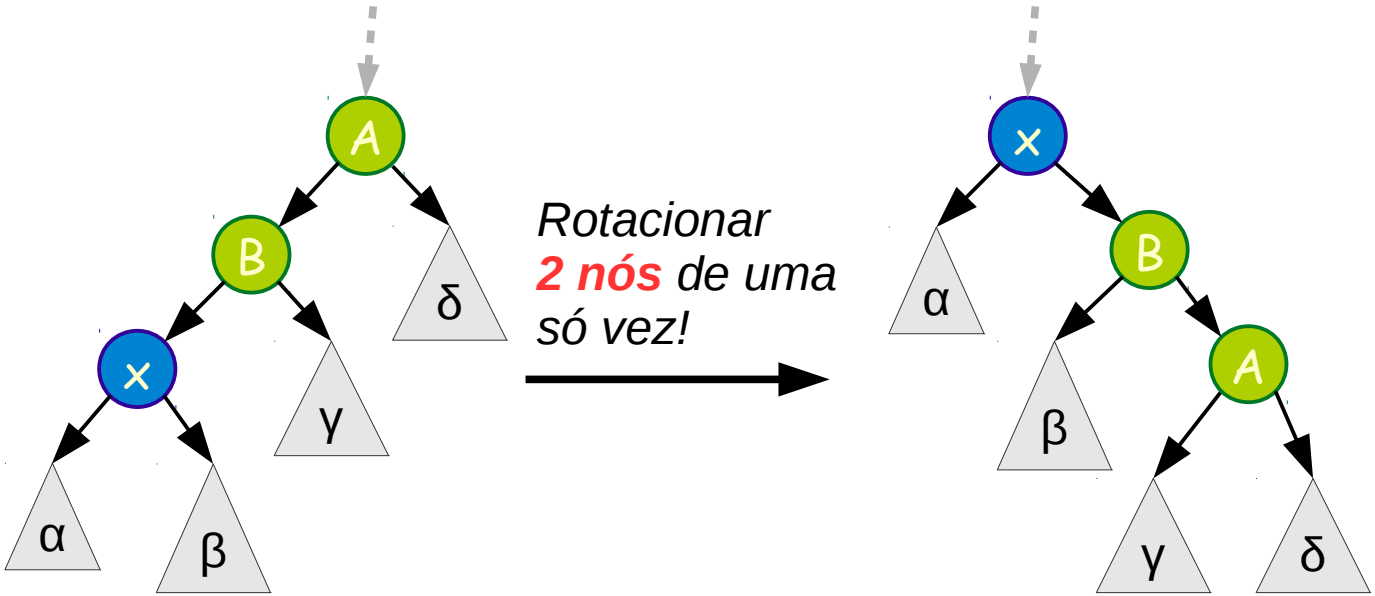
- Rotaciona na direção da raiz.

Caso 3: *zig-zig* (pai de x na mesma direção do avô)



Procedimento

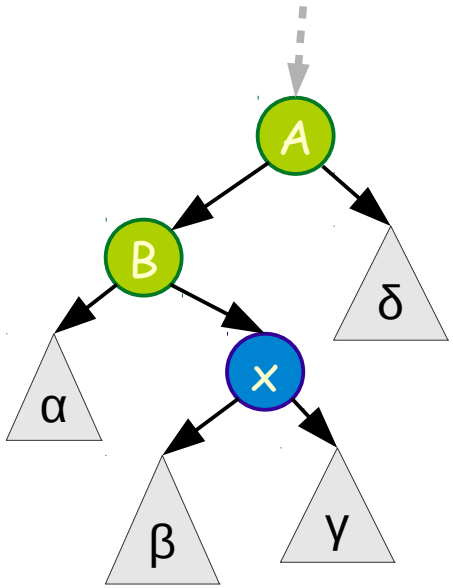
Caso 3: *zig-zig* (pai de x na mesma direção do avô)



Procedimento

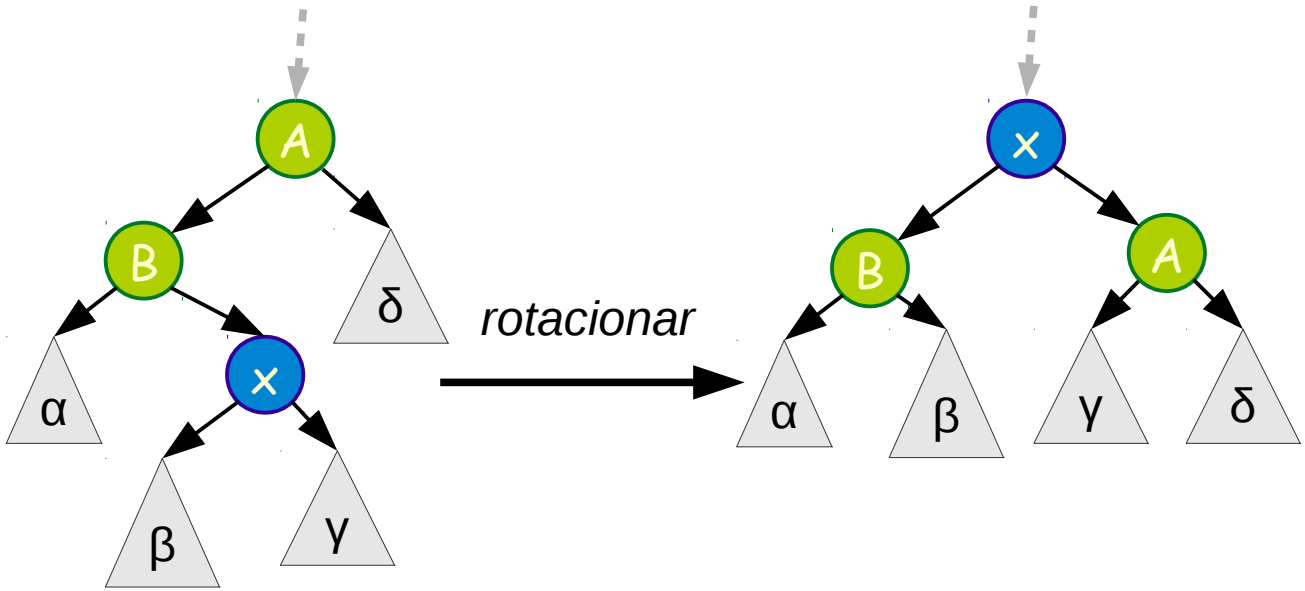
- Rotaciona avô e depois pais na mesma direção.

Caso 4: *zig-zag* (pai de x na mesma oposta à do avô)



Procedimento

Caso 4: *zig-zag* (pai de x na mesma oposta à do avô)



Procedimento

- Rotaciona pai na direção do pai e depois avô na direção oposta.

Implementação em C

Afunilar

```
void afunilar(NoArv **r, NoArv *x) {
    // enquanto não chega na raiz
    while( *r != x ) {
        // caso zig
        if (x->pai == *r) {
            ...
        }
```

Implementação em C

Afunilar

```
void afunilar(NoArv **r, NoArv *x) {
    // enquanto não chega na raiz
    while( *r != x ) {
        // caso zig
        if (x->pai == *r) {
            ...
        } else {
            NoArv *pai = x->pai, *avo = pai->pai;
```

Implementação em C

Afunilar

```
void afunilar(NoArv **r, NoArv *x) {
    // enquanto não chega na raiz
    while( *r != x ) {
        // caso zig
        if (x->pai == *r) {
            ...
        } else {
            NoArv *pai = x->pai, *avo = pai->pai;
            // caso zig-zig
            if ((x==pai->esq && pai==avo->esq) ||
                (x==pai->dir && pai==avo->dir) )  {
                ...
```

Implementação em C

Afunilar

```
void afunilar(NoArv **r, NoArv *x) {
    // enquanto não chega na raiz
    while( *r != x ) {
        // caso zig
        if (x->pai == *r) {
            ...
        } else {
            NoArv *pai = x->pai, *avo = pai->pai;
            // caso zig-zig
            if ((x==pai->esq && pai==avo->esq) ||
                (x==pai->dir && pai==avo->dir) )  {
                ...
            // caso zig-zag
            } else {
                ...
            }
        }
    }
}
```

Operações de Conjunto

Operações básicas

Buscar:

1. Inserir em árvore de busca
2. Se encontrou, afunilar elemento acessado

Operações de Conjunto

Operações básicas

Buscar:

1. Inserir em árvore de busca
2. Se encontrou, afunilar elemento acessado

Inserir:

1. Inserir em árvore de busca
2. Afunilar elemento inserido

Operações de Conjunto

Operações básicas

Buscar:

1. Inserir em árvore de busca
2. Se encontrou, afunilar elemento acessado

Inserir:

1. Inserir em árvore de busca
2. Afunilar elemento inserido

Remover:

1. Buscar elemento e guardar o pai se houver
2. Remover elemento
3. Se houver pai, afunilar o pai

Outras operações

Suponha que todos os elementos de S são menores que os elementos de T.

Outras operações

Suponha que todos os elementos de S são menores que os elementos de T.
Como criar uma árvore com a união?

Outras operações

Suponha que todos os elementos de S são menores que os elementos de T.
Como criar uma árvore com a união?

Juntar(S, T)

1. Encontrar nó x que é máximo de S
2. Afunilar o nó x
3. Fazer T ser filho direito de x

Outras operações

Suponha que todos os elementos de S são menores que os elementos de T.
Como criar uma árvore com a união?

Juntar(S, T)

1. Encontrar nó x que é máximo de S
2. Afunilar o nó x
3. Fazer T ser filho direito de x

Suponha que temos um número n uma árvore T.

Outras operações

Suponha que todos os elementos de S são menores que os elementos de T. Como criar uma árvore com a união?

Juntar(S, T)

1. Encontrar nó x que é máximo de S
2. Afunilar o nó x
3. Fazer T ser filho direito de x

Suponha que temos um número n uma árvore T. Como separar elementos menores ou iguais n dos maiores que n?

Outras operações

Suponha que todos os elementos de S são menores que os elementos de T. Como criar uma árvore com a união?

Juntar(S, T)

1. Encontrar nó x que é máximo de S
2. Afunilar o nó x
3. Fazer T ser filho direito de x

Suponha que temos um número n uma árvore T. Como separar elementos menores ou iguais n dos maiores que n?

Separar(T, n):

1. Buscar nó x que é o maior elemnto menor ou igual a n
2. Afunilar x
3. Fazer T o filho direito de x
4. Fazer S a árvore remanescente.

Exercício

1. A ordem das rotações nos passos *zig-zig* e *zig-zag* é importante para manter o bom balanceamento da árvore. Em cada caso, quem deve ser rotacionado primeiro: pai ou avô?
2. Desenhe a árvore splay-tree que é obtida a partir de uma árvore vazia após: a inserção de elementos 1, 3, 5, 7, 8, busca do elemento 3 e busca do elemento 8.