Estruturas de Dados

Árvores Binárias de Busca

Aula 07

Prof. Felipe A. Louza

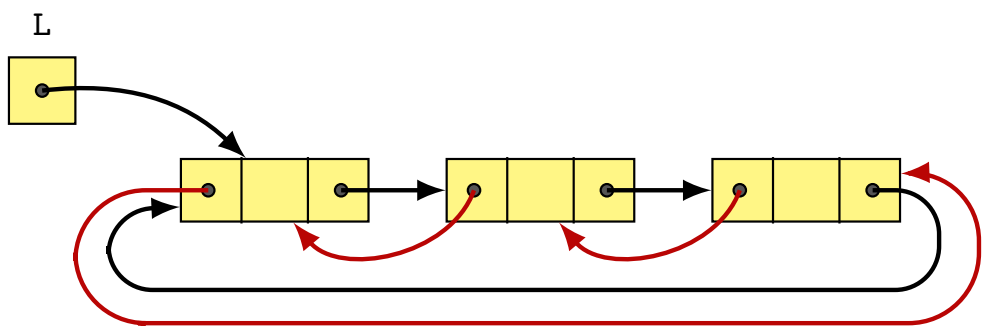
Roteiro

Roteiro

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Ligadas:

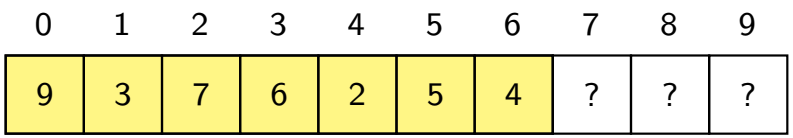
- Podemos **inserir** e **remover** em $\mathrm{O}(1)$
- Mas **buscar** demora $\mathrm{O}(n)$



Tempo para Inserção, Remoção e Busca

Se usarmos vetores (não-ordenados):

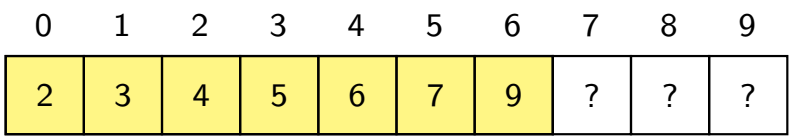
- Podemos **inserir** e **remover** em $\mathrm{O}(1)$
  - insire no final
  - para remover, troque com o último e remova o último
- Mas **buscar** demora $\mathrm{O}(n)$



Tempo para Inserção, Remoção e Busca

Por fim, se usarmos vetores ordenados:

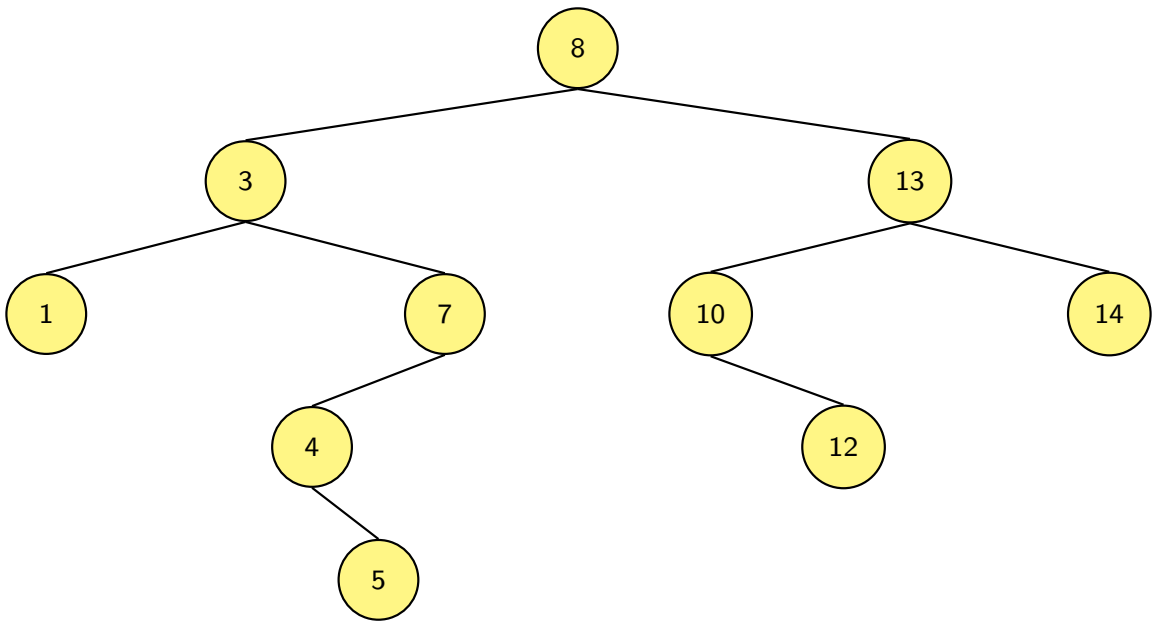
- Podemos **buscar** em $\mathrm{O}(\lg n)$
- Mas **inserir** e **remover** leva $\mathrm{O}(n)$



Tempo para Inserção, Remoção e Busca

Veremos agora árvores binárias de busca

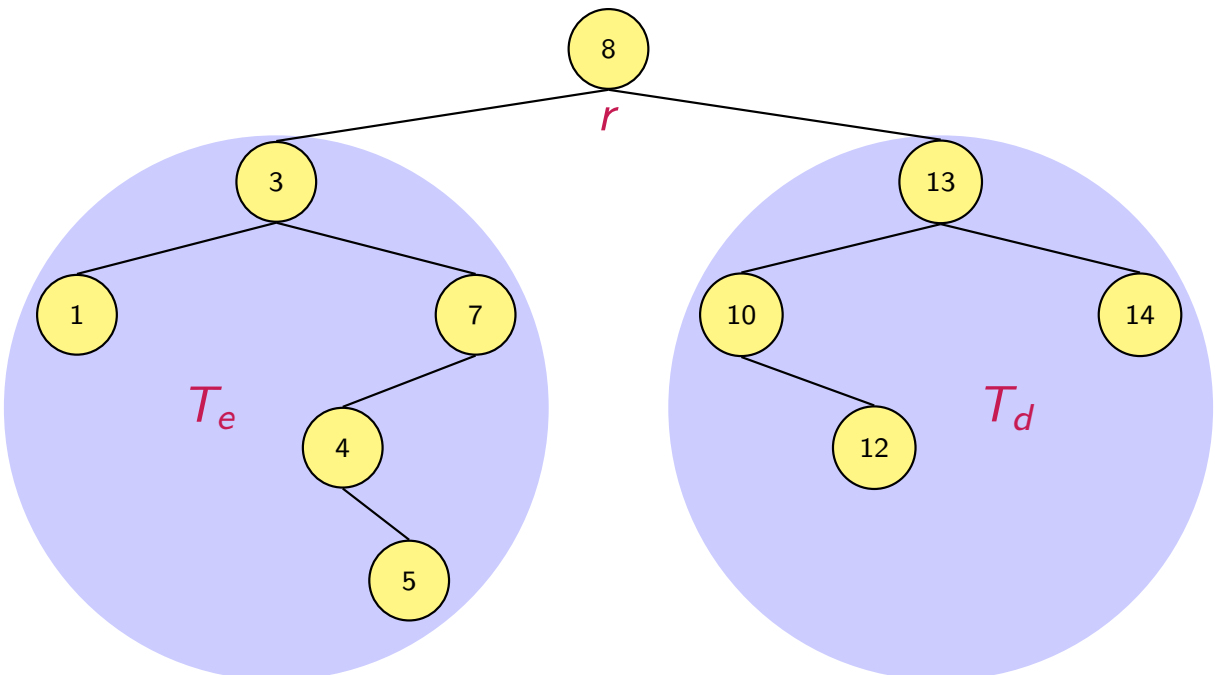
- primeiro uma versão simples,
- depois uma versão sofisticada: três operações levam $\mathrm{O}(\lg n)$



Árvore Binária de Busca

Uma Árvore Binária de Busca (ABB) é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável

- Cada nó r, com subárvores esquerda T_e e direita T_d satisfaz a seguinte propriedade:
  1. $e < r$ para todo elemento $e \in T_e$
  2. $d > r$ para todo elemento $d \in T_d$



TAD - Interface

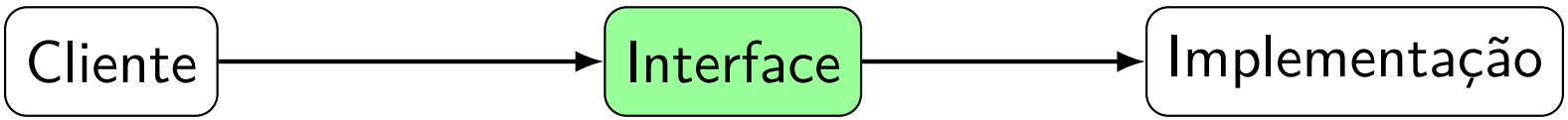


abb.h

```
1  #ifndef ABB_H
2  #define ABB_H
3
4  //Dados
5  typedef struct no {
6      int chave;
7      struct no *esq, *dir, *pai;
8  } No;
9
10 //Funções
11 No* criar_arvore();
12 void destruir_arvore(No **p);
```

```
14 void imprimir_arvore(No *p, int h);
15 void imprimir_inordem(No *p);
16
17 No*  inserir(No *p, int chave);
18 void remover(No **p, int chave);
19
20 No* buscar(No *p, int chave);
21
22 #endif
```

ABB - Criar e destruir árvore

abb.c

```
5  No* criar_arvore(){
6    return NULL;
7  }
```

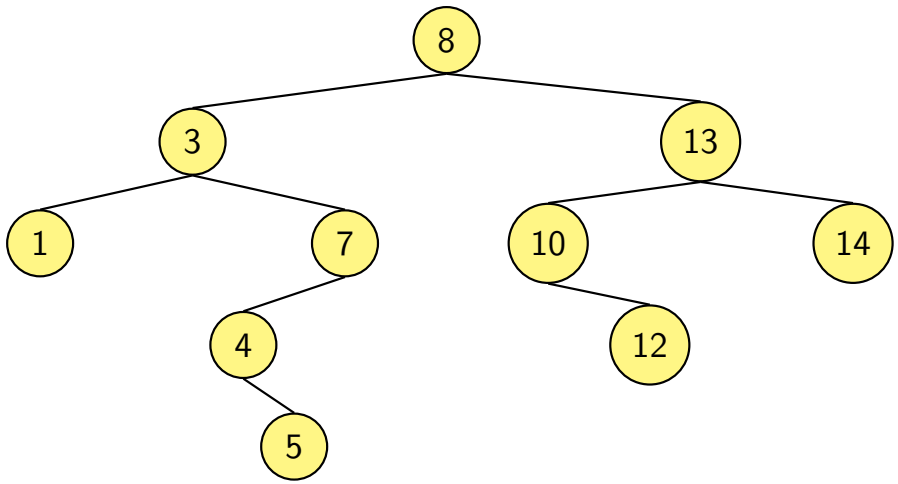
```
9  void destruir_arvore(No **p){//função recursiva
10   if (*p == NULL) return;
11   destruir_arvore(&((*p)->esq));
12   destruir_arvore(&((*p)->dir));
13   free(*p);
14   *p = NULL;
15 }
```

Código do cliente: `No *T = criar_arvore();`

ABB - Imprimir árvore

abb.c

```
17 void imprimir_arvore(No *p, int h) {
18   int i;
19   if (p != NULL) {
20     imprimir_arvore(p->dir, h+1);
21     for (i = 0; i < h; i++)
22       printf("-");
23     printf(" %d\n", p->chave);
24     imprimir_arvore(p->esq, h+1);
25   }
26 }
```




```
28 void imprimir_inordem(No *p) {
29   if (p != NULL) {
30     imprimir_inordem(p->esq);
31     printf("%d ", p->chave);
32     imprimir_inordem(p->dir);
33   }
34 }
```

Cliente

exemplo1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "abb.h"

int main() {
  int i, v[10] = {8, 3, 1, 7, 13, 10, 14, 12, 4, 5};
  No *T = criar_arvore();
  for (i = 0; i < 10; i++)
    T = inserir(T, v[i]);
  imprimir_arvore(T, 0);
  imprimir_inordem(T);
  printf("\n");
  destruir_arvore(&T);
return 0;
}
```

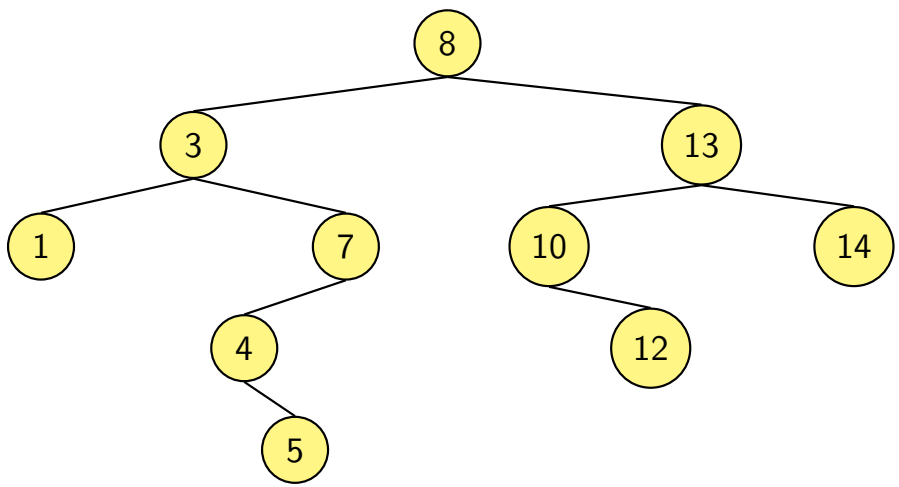
Makefile

Vamos usar o Makefile para compilar:

```
1 exemplo1: exemplo1.c abb.o
2   gcc $^ -o $@
```

Vamos executar:

```
1  $ ./exemplo1
2  -- 14
3  - 13
4  --- 12
5  -- 10
6   8
7  -- 7
8  ---- 5
9  --- 4
10 - 3
11 -- 1
12 1 3 4 5 7 8 10 12 13 14
```



Roteiro

ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

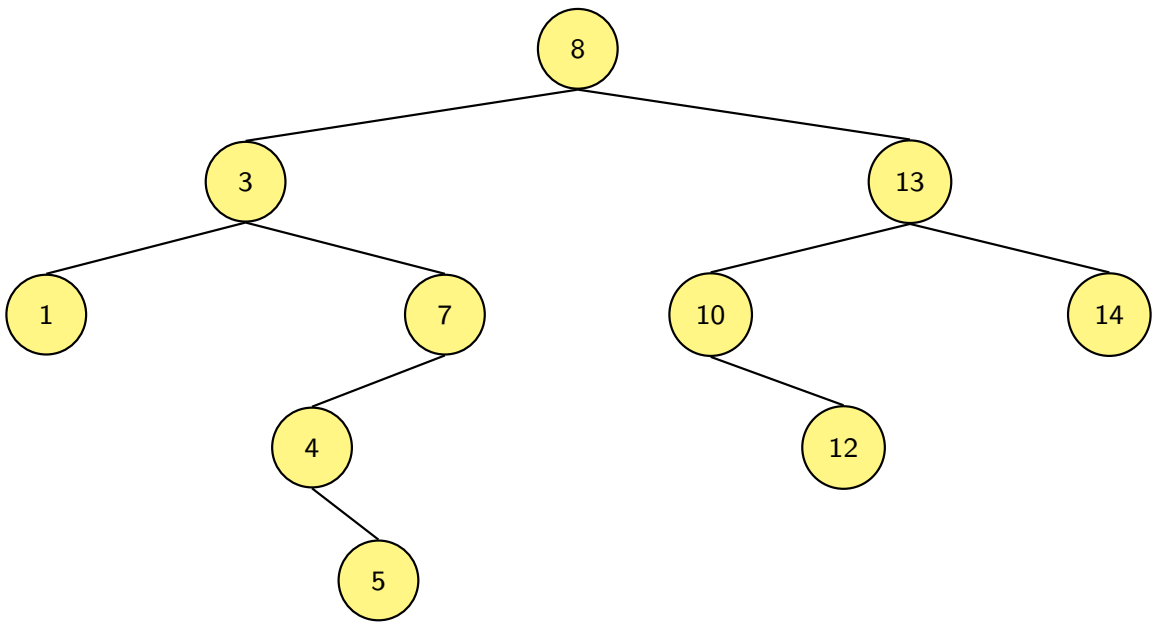


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

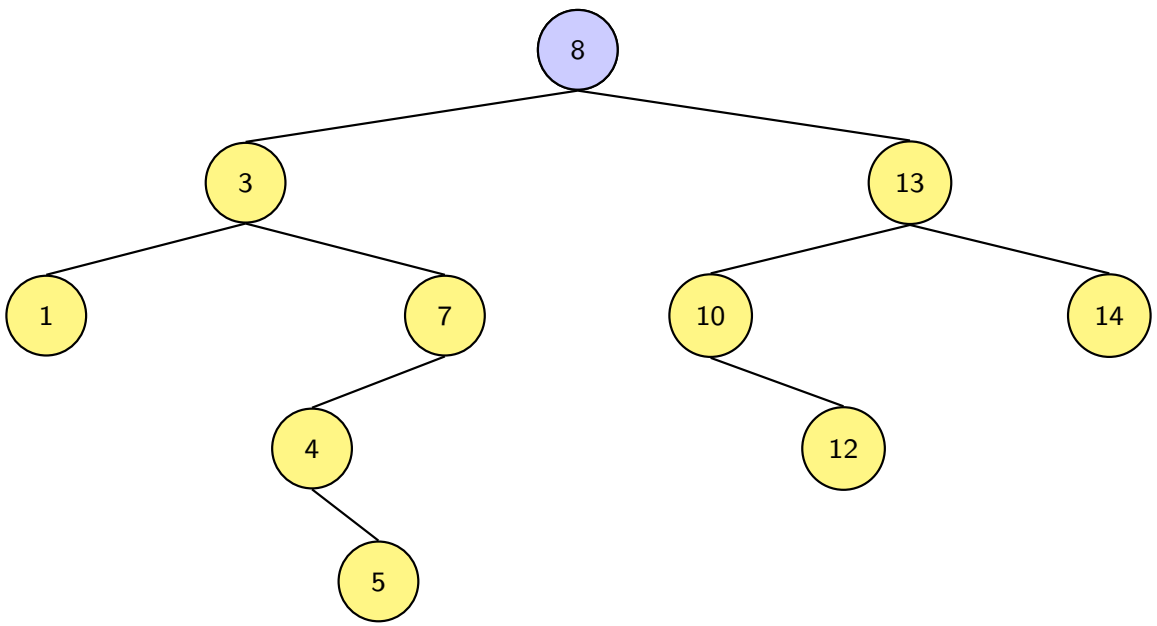


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

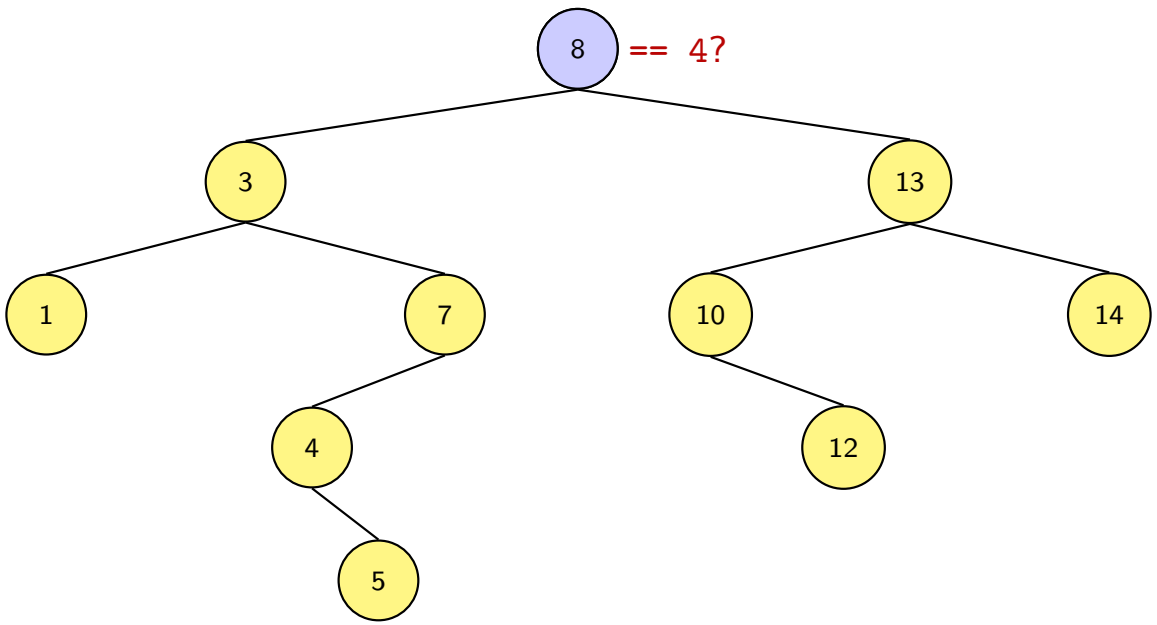


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

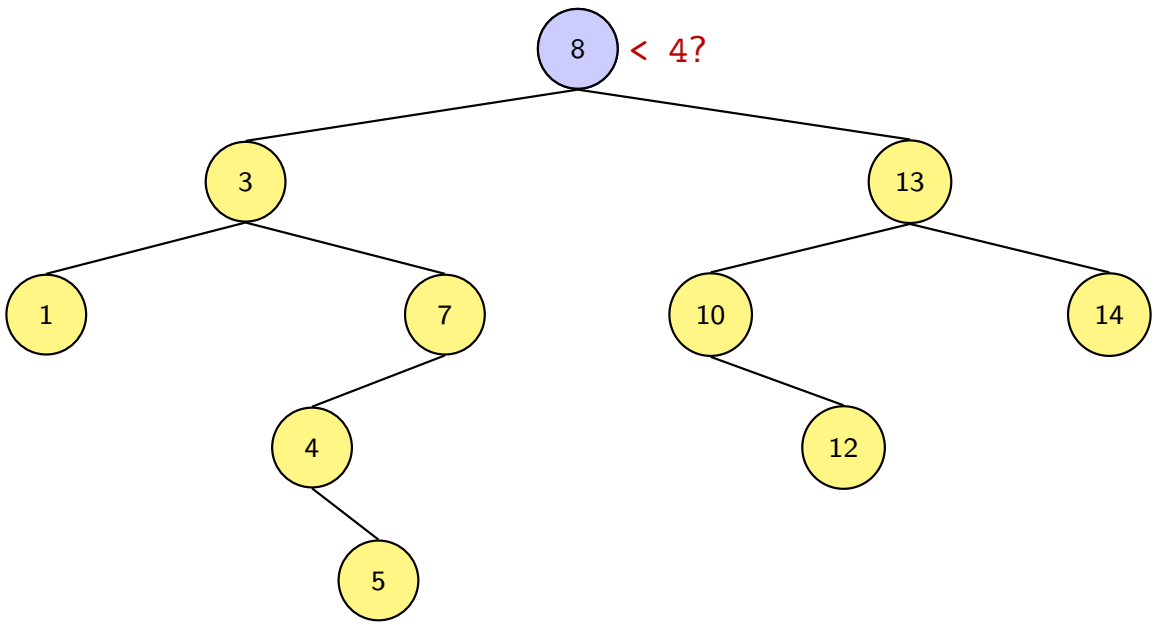


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

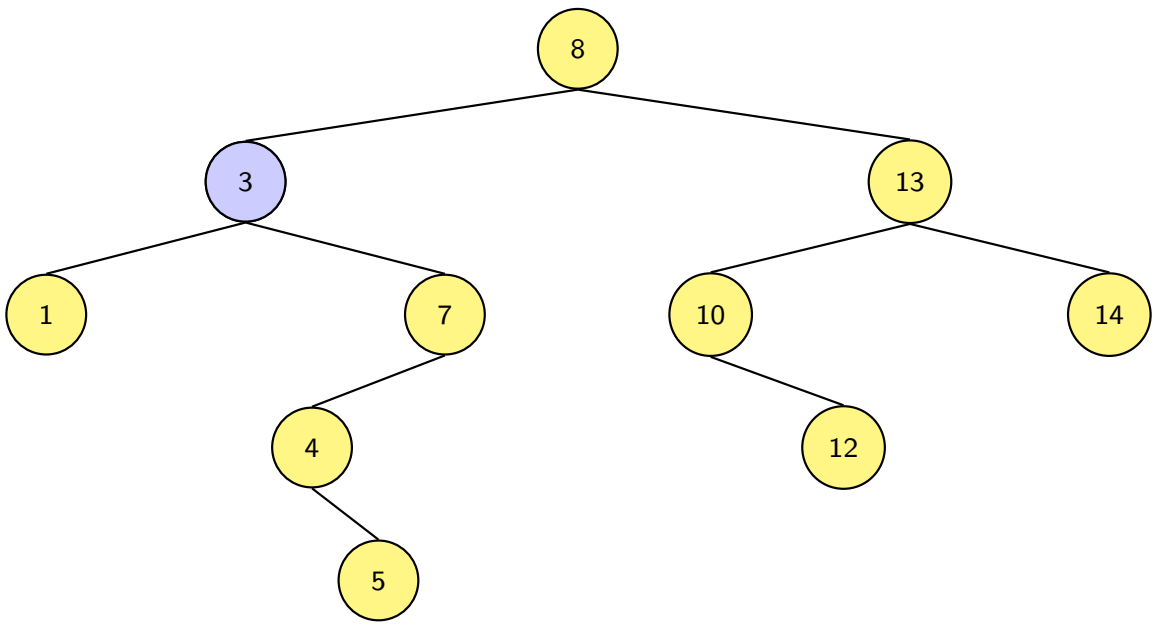


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

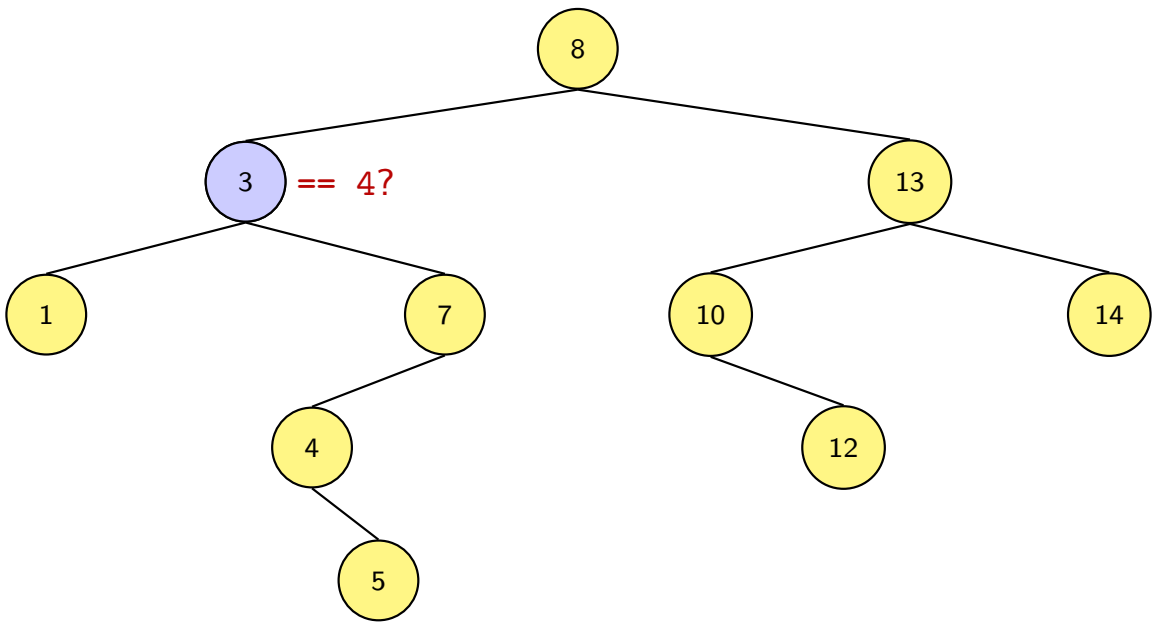


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

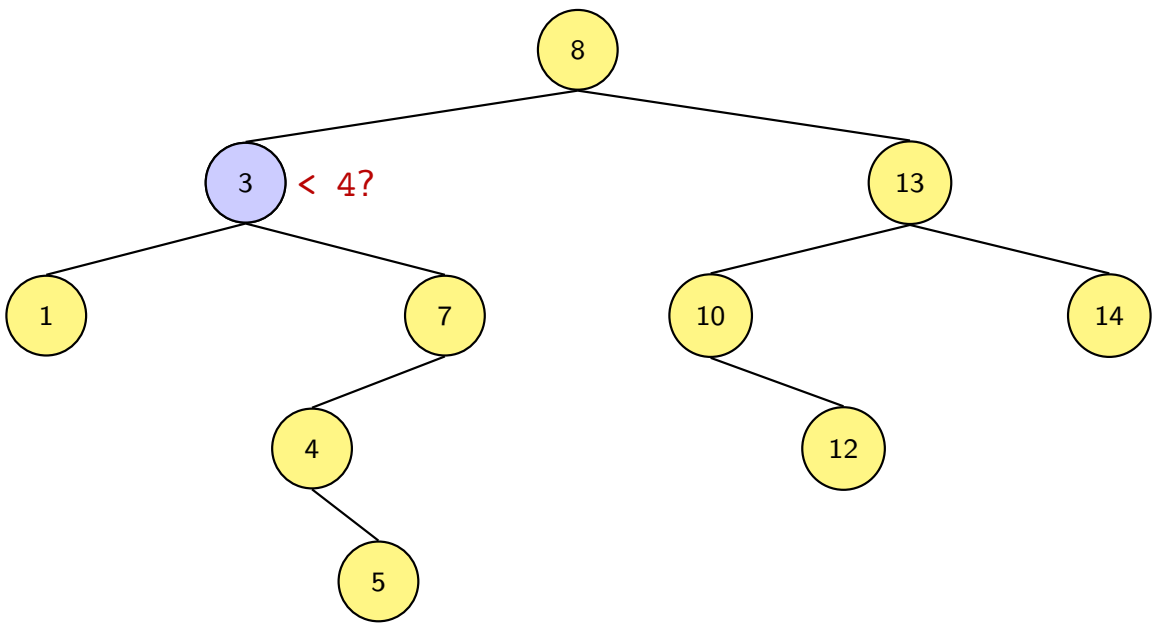


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

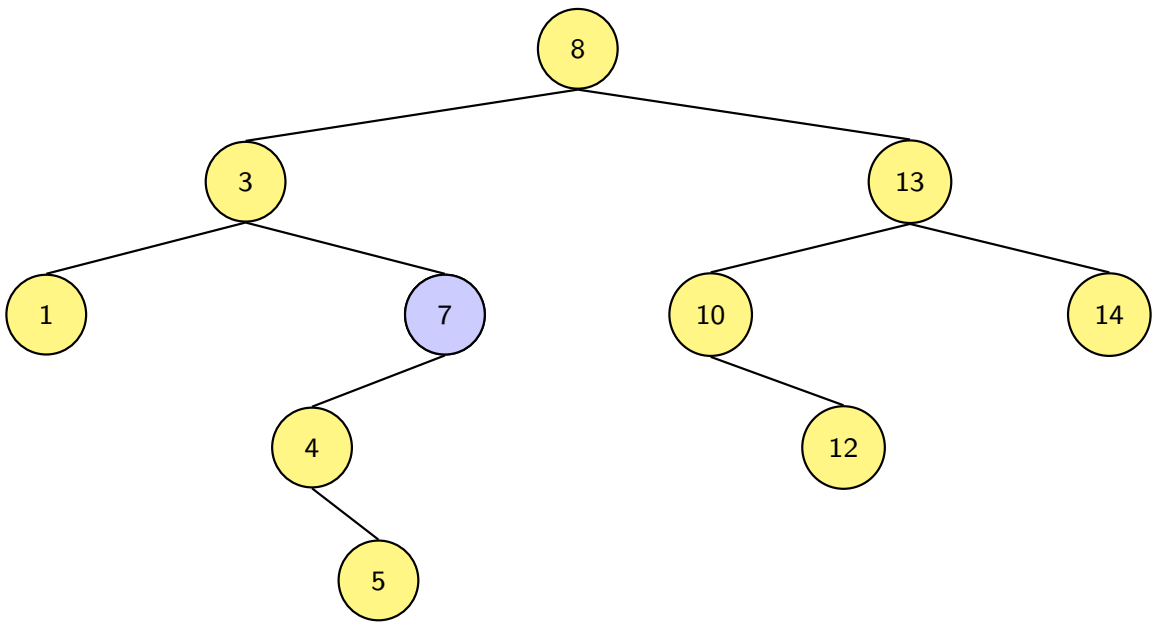


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

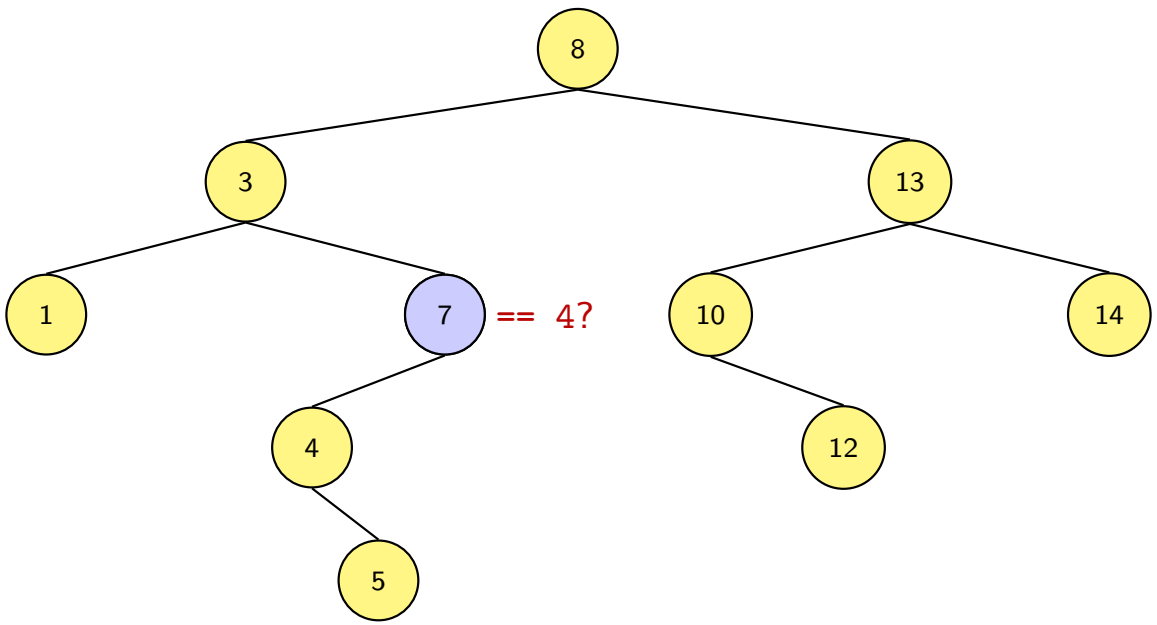


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

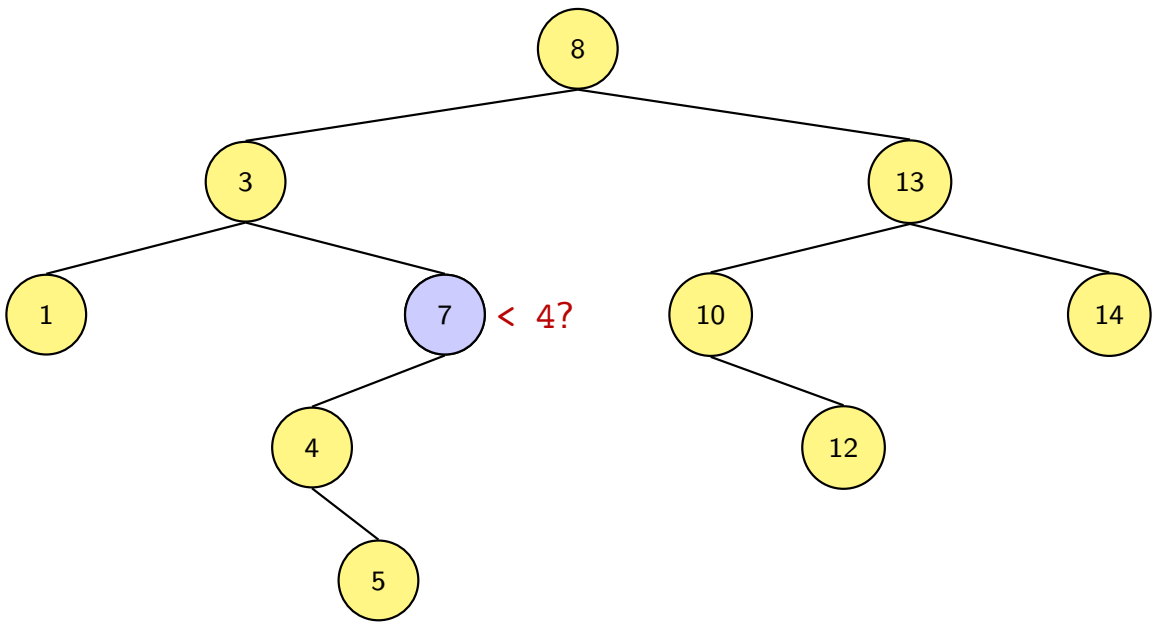


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

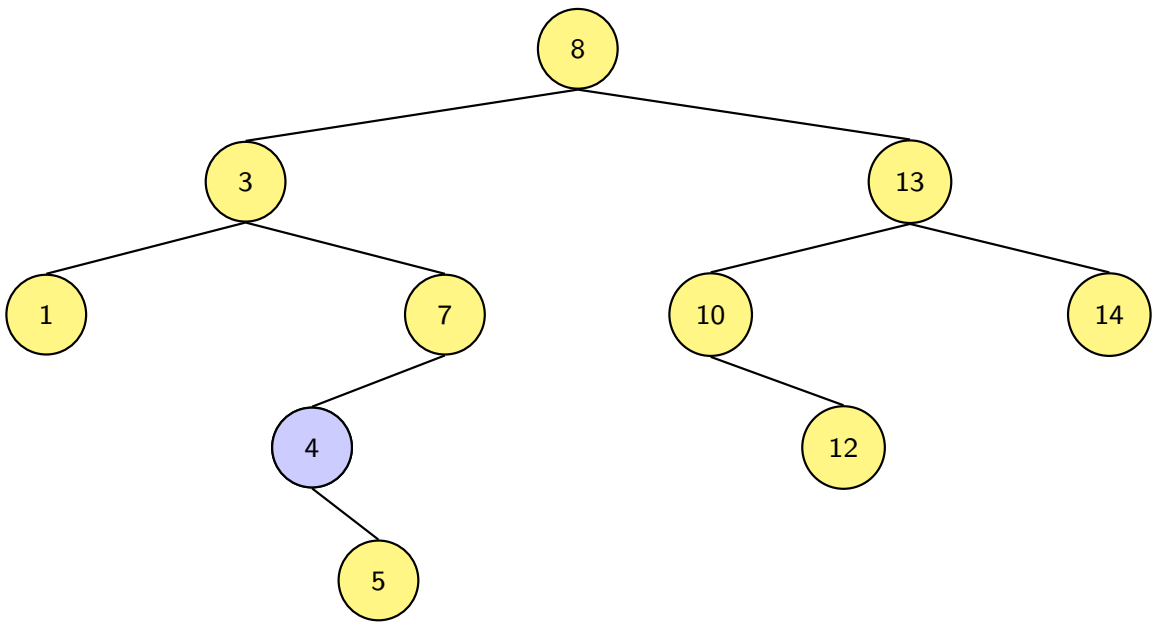


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

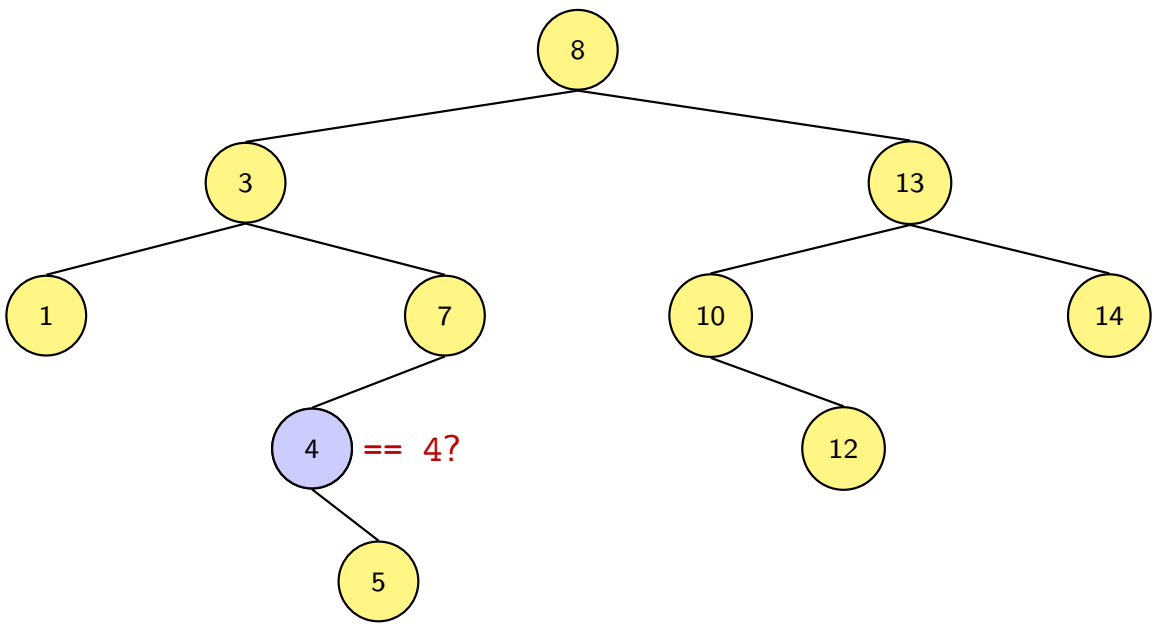


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 4

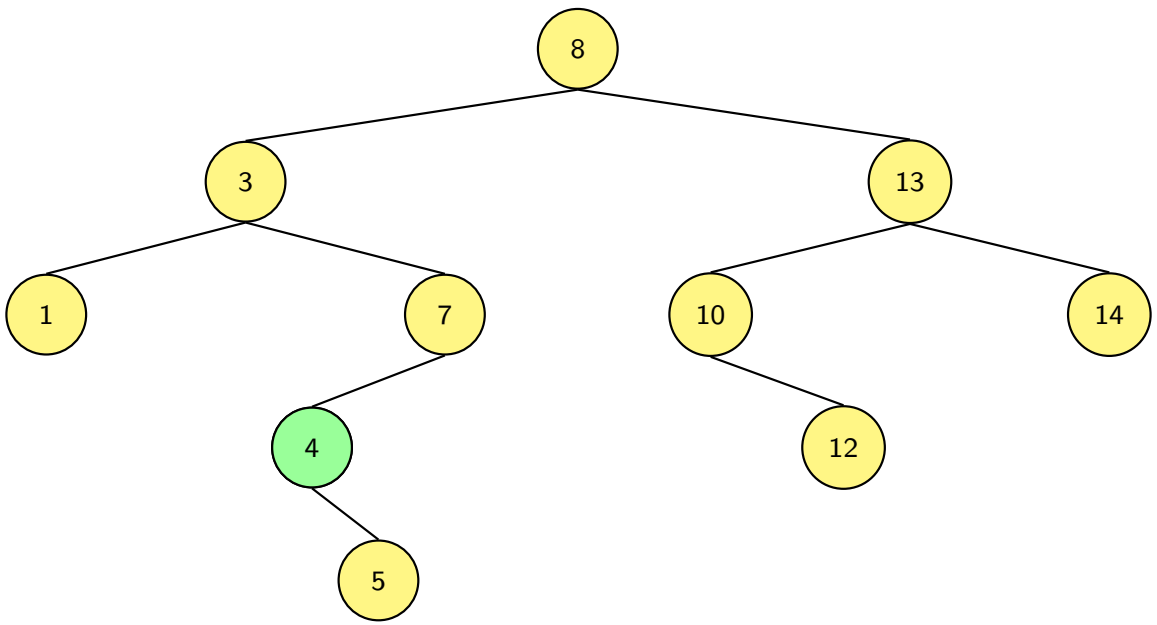


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

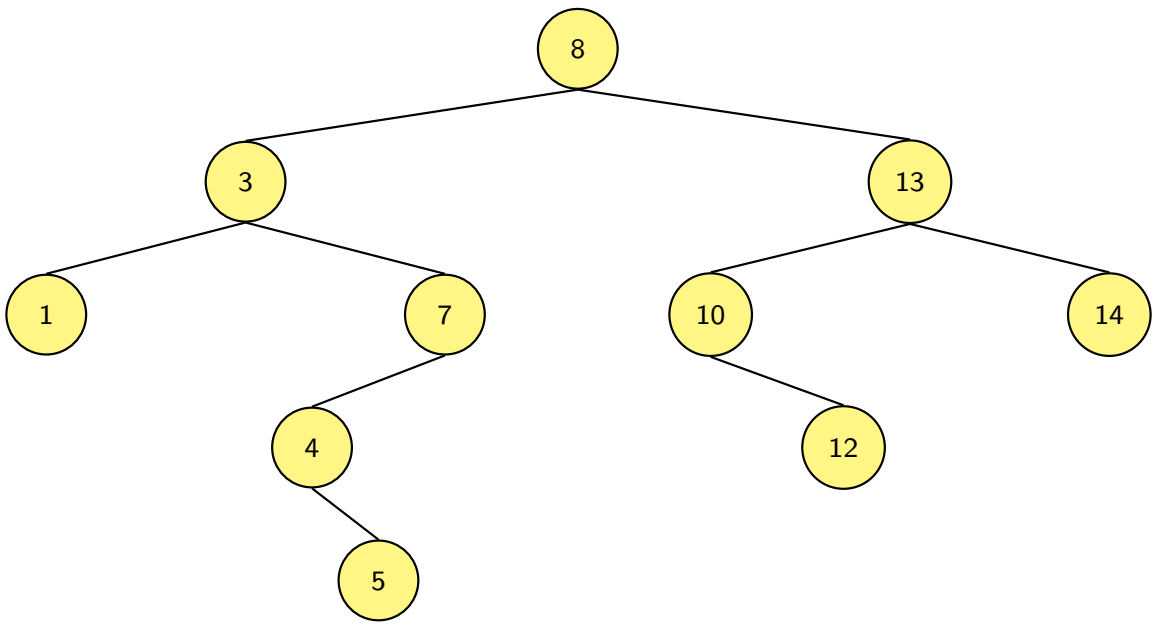


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

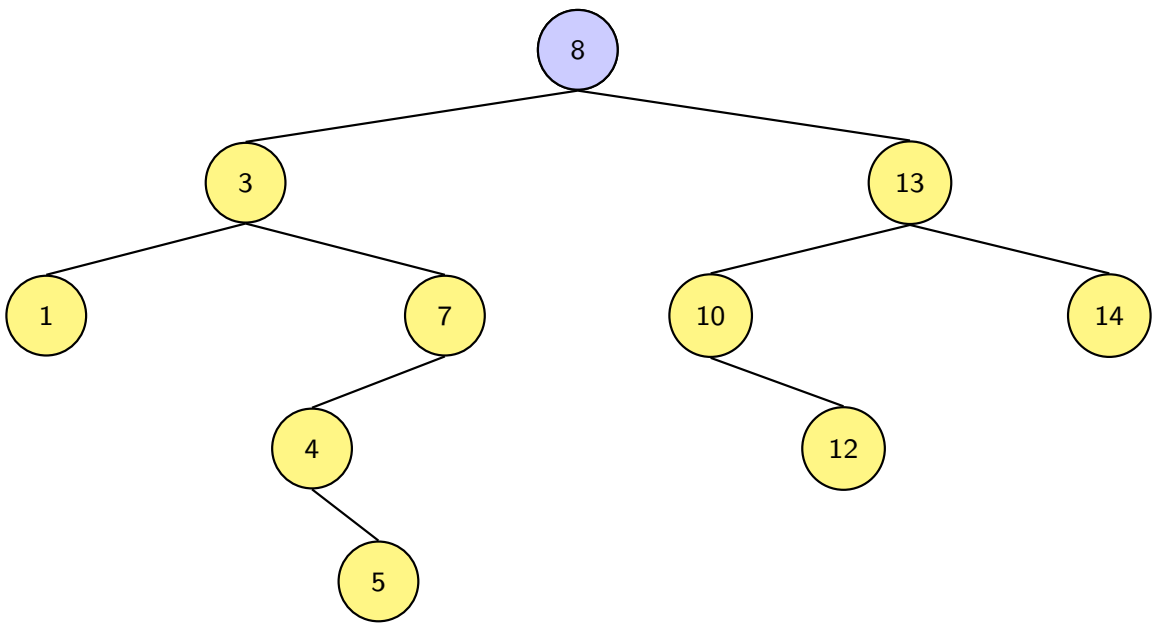


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

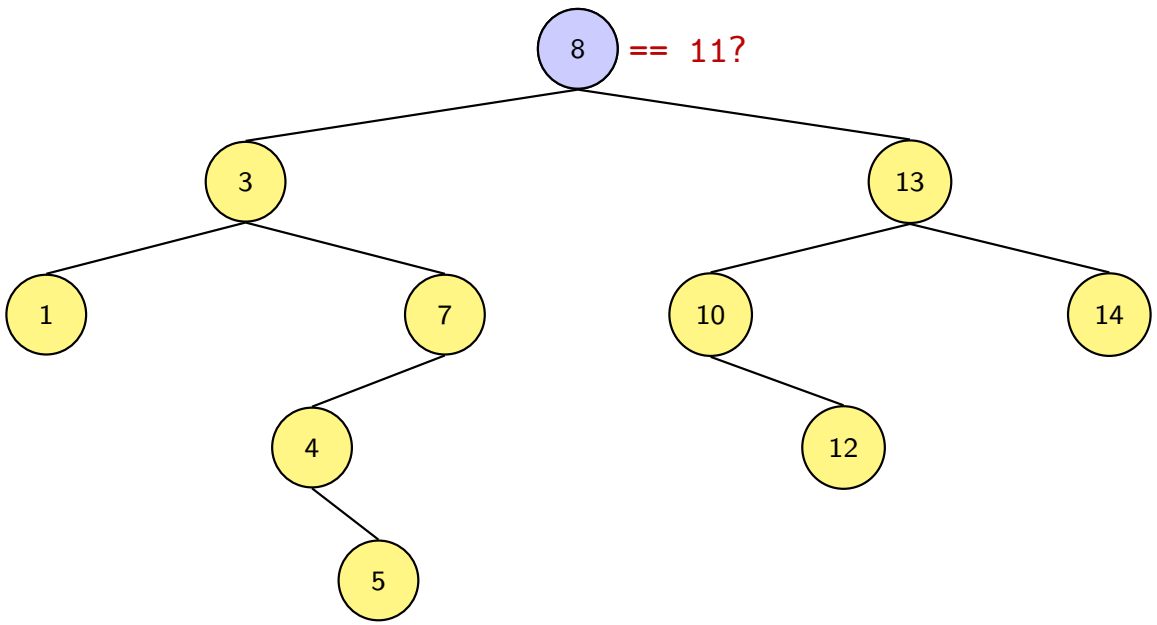


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

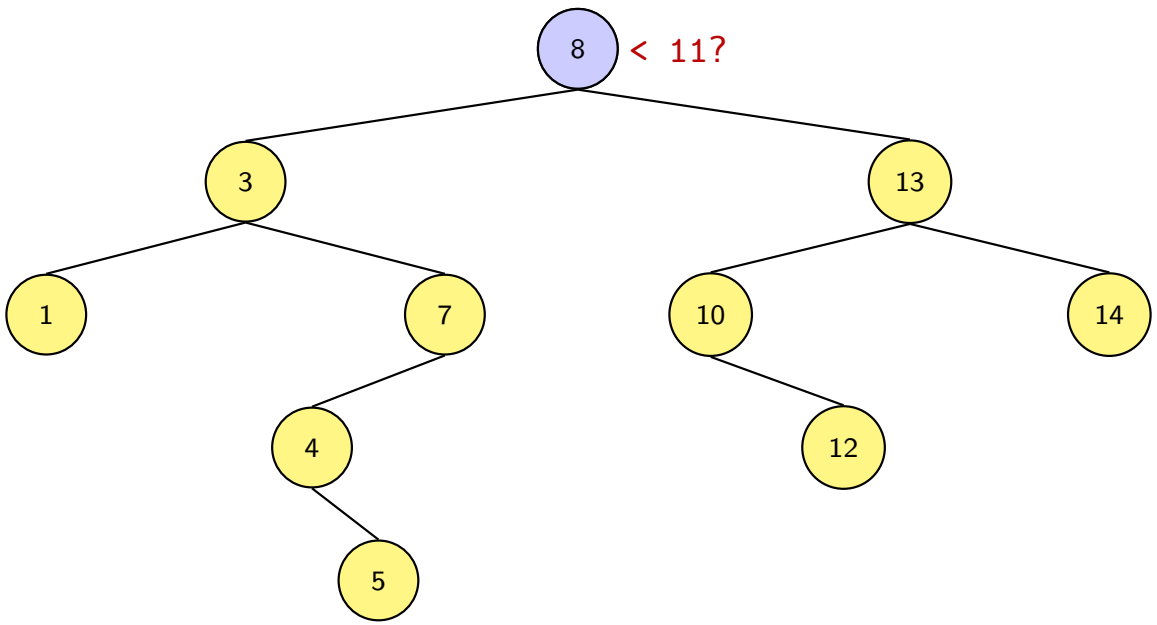


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

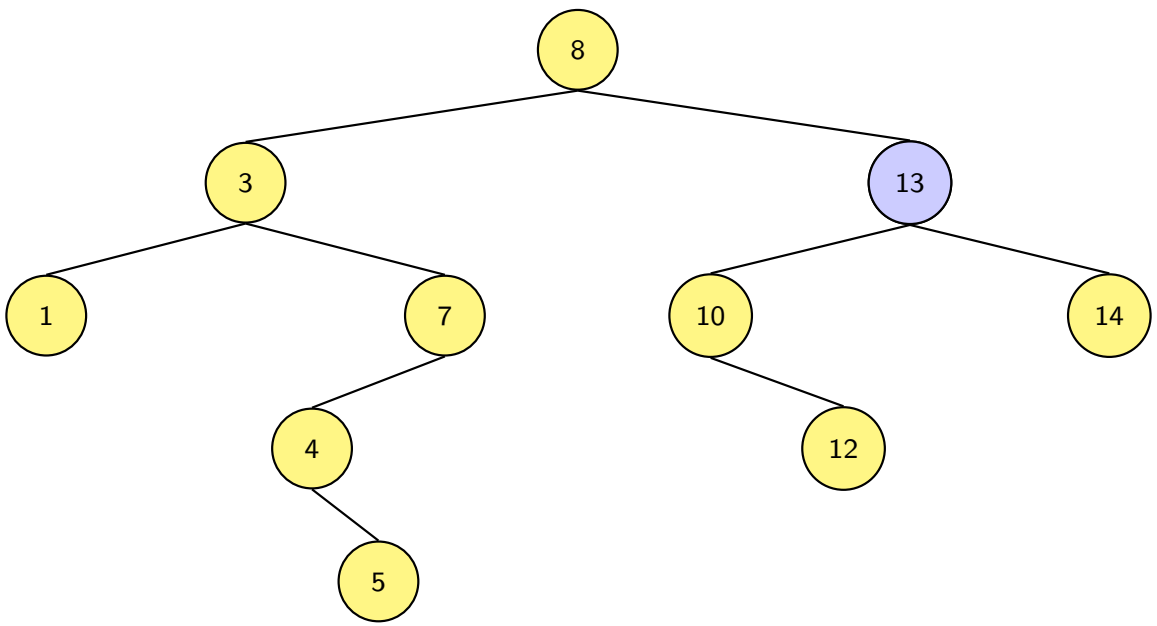


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

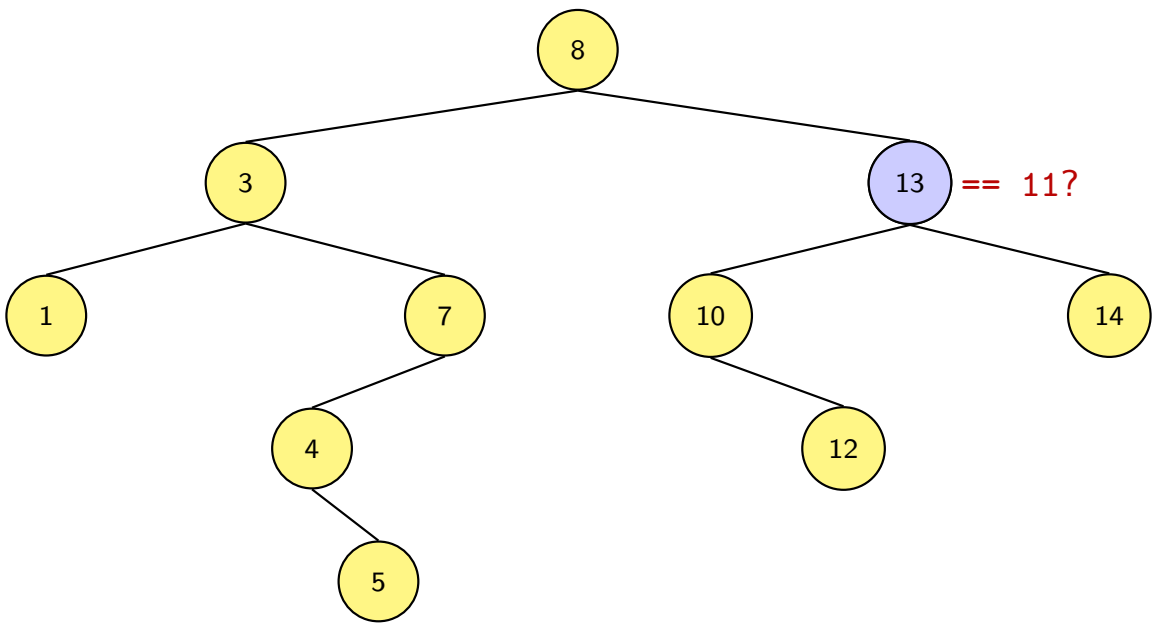


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

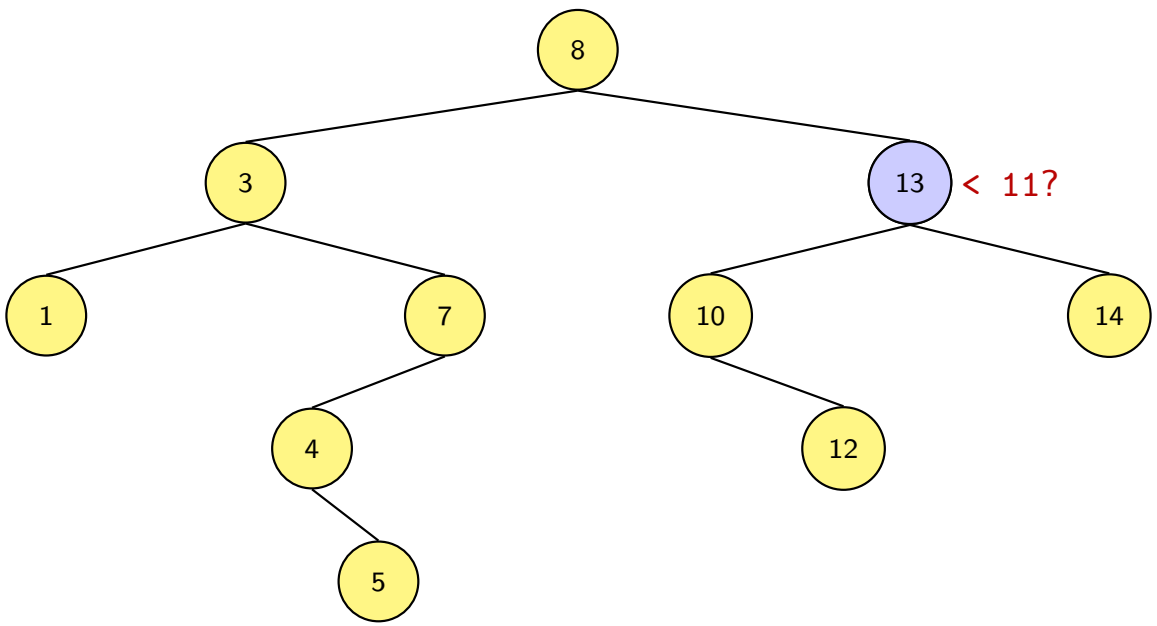


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

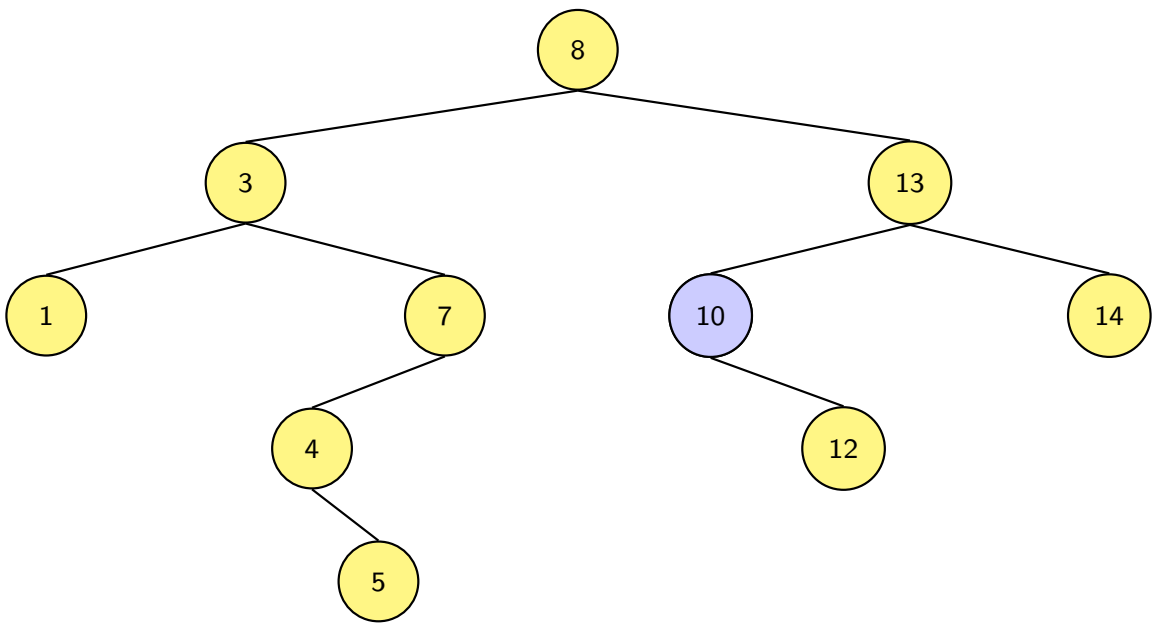


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

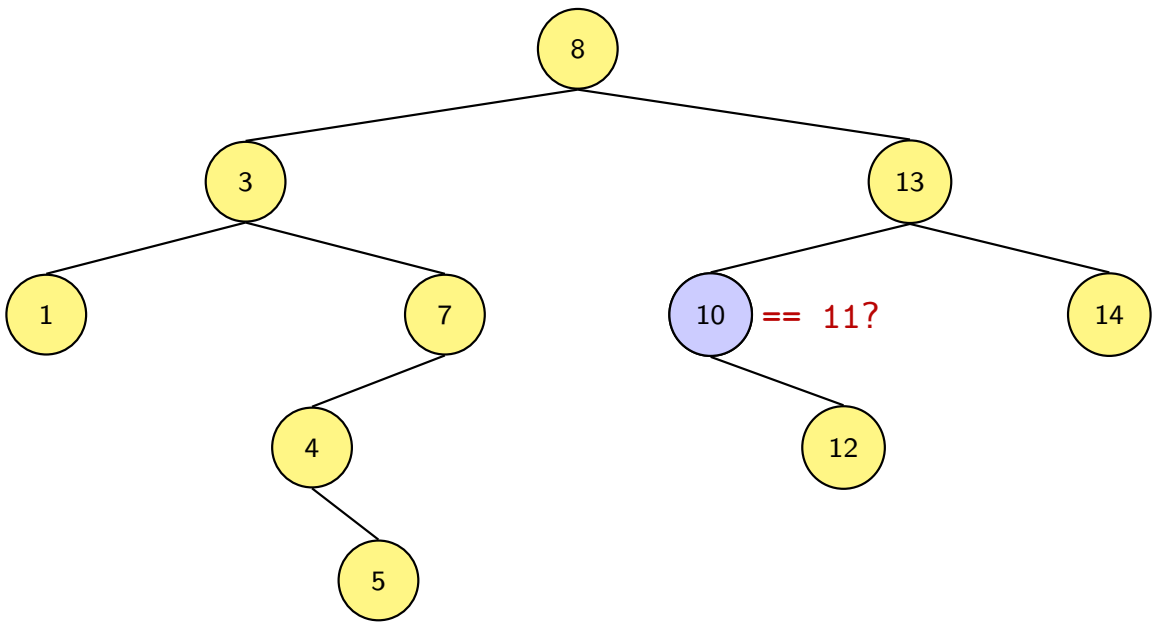


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

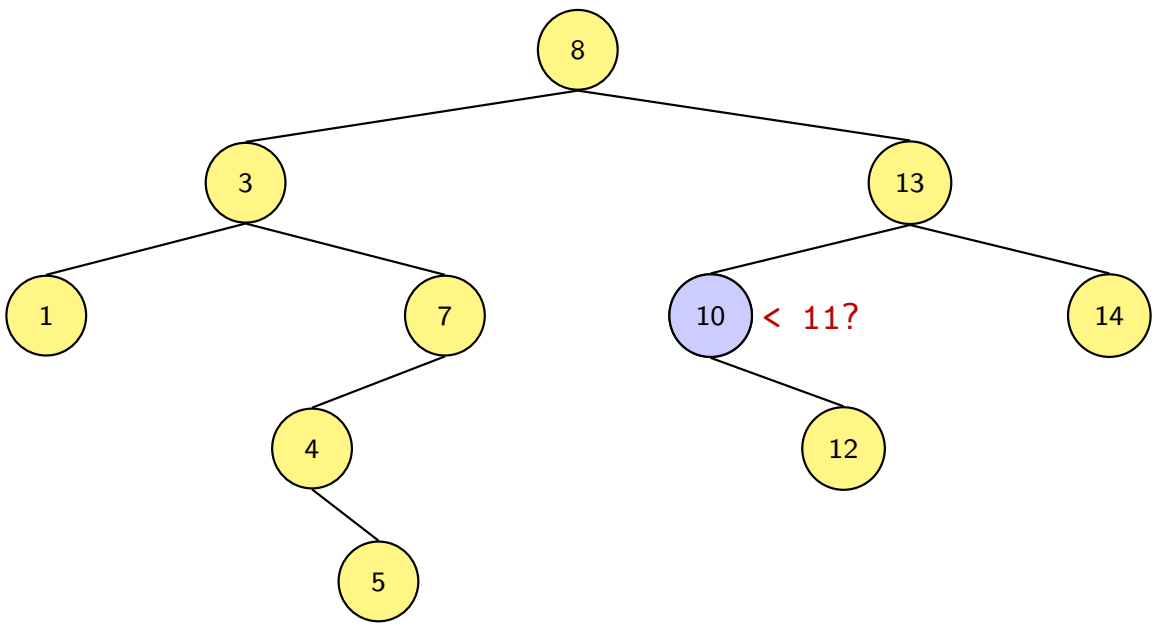


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

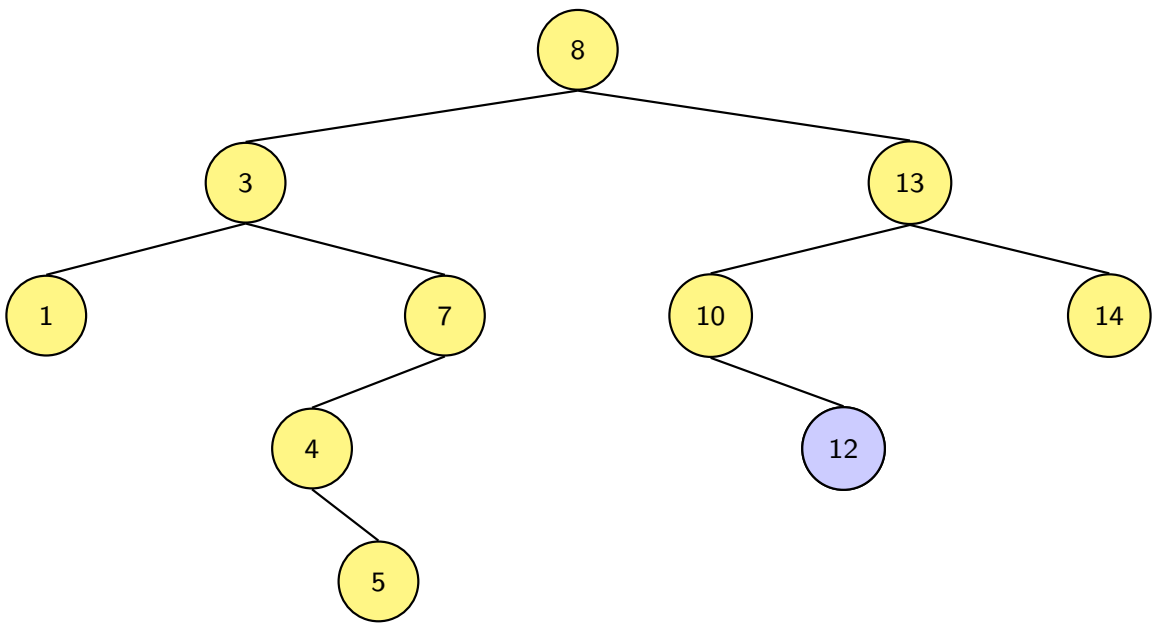


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

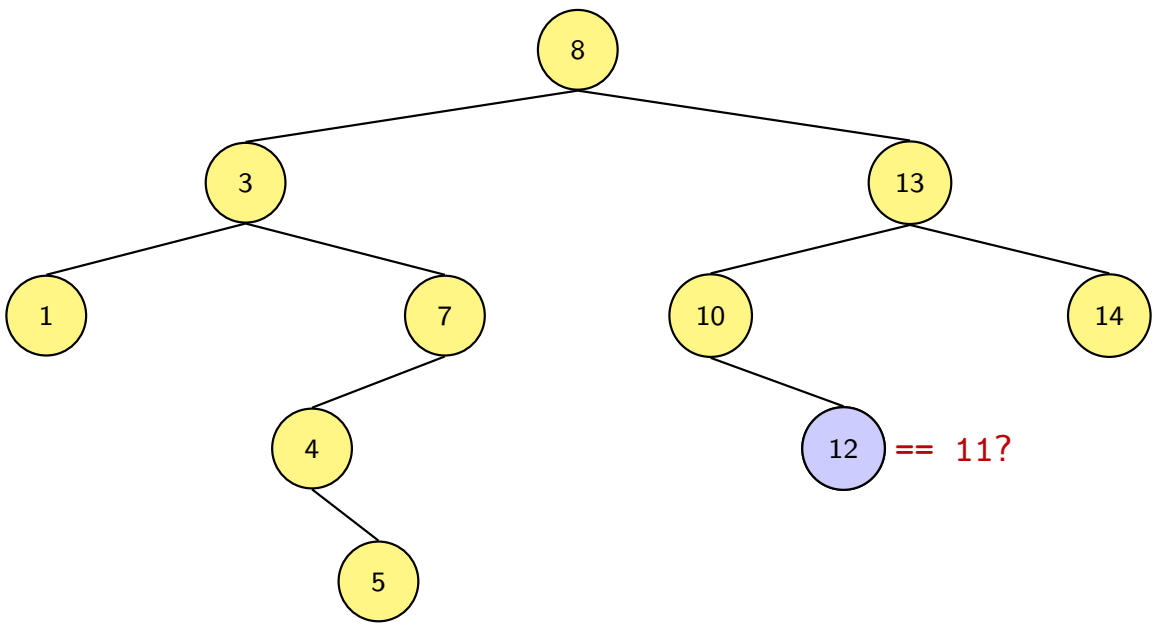


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

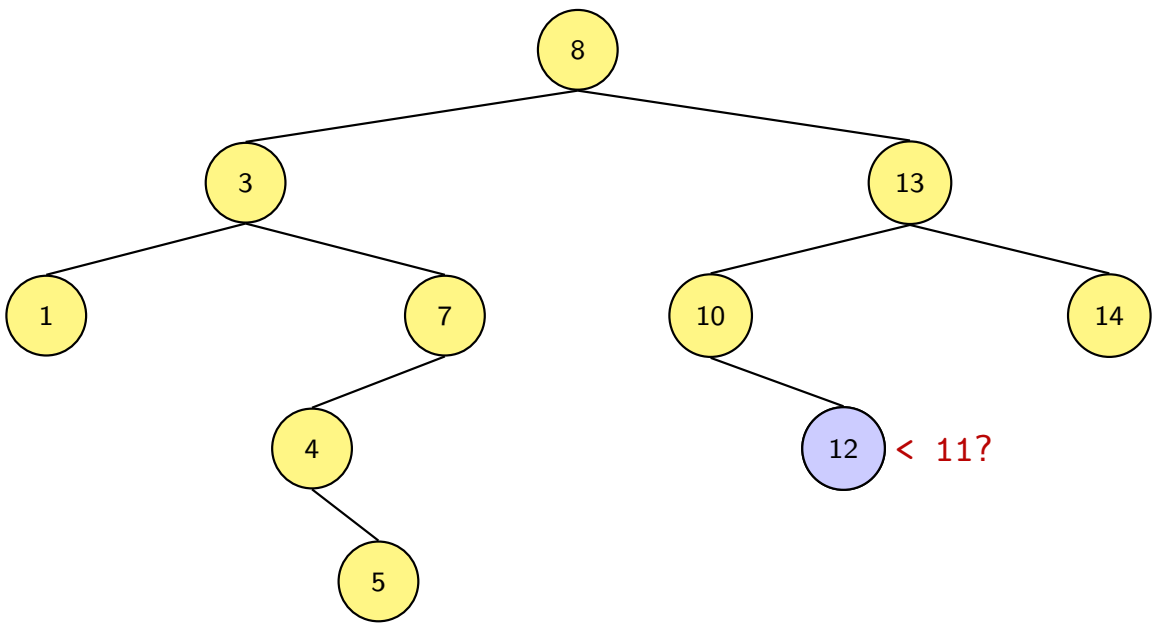


ABB - Busca por um valor

A ideia é semelhante àquela da busca binária:

- Ou o **valor** a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na **subárvore esquerda**
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na **subárvore direita**

Ex: Buscando por 11

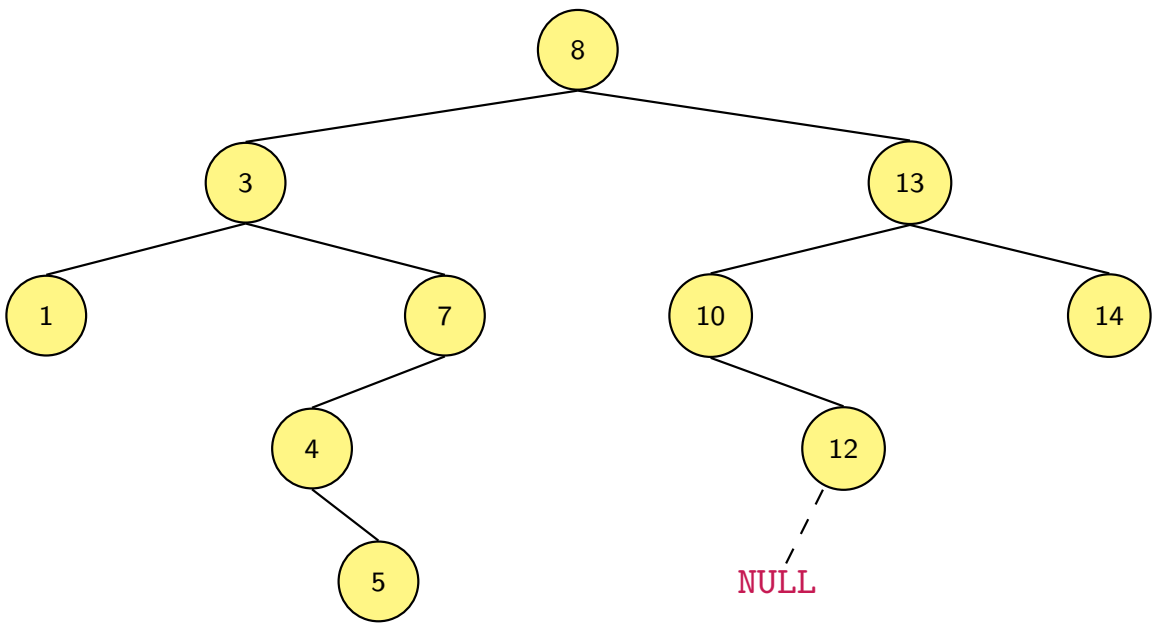
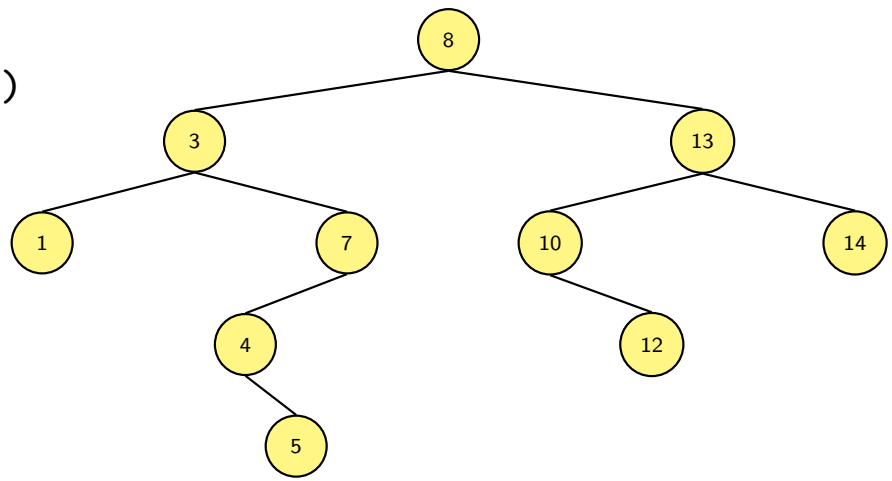


ABB - Busca por um valor

abb.c

```
36 No* buscar(No *p, int chave) {
37   if (p == NULL || chave == p->chave)
38     return p;
39   if (chave < p->chave)
40     return buscar(p->esq, chave);
41   else
42     return buscar(p->dir, chave);
43 }
44
```



- A função retorna o endereço do nó em que encontrou a chave
- Caso contrário, retorna NULL

Eficiência da busca

Qual é o custo computacional da busca?

Eficiência da busca

Qual é o custo computacional da busca?

- depende da forma da árvore...

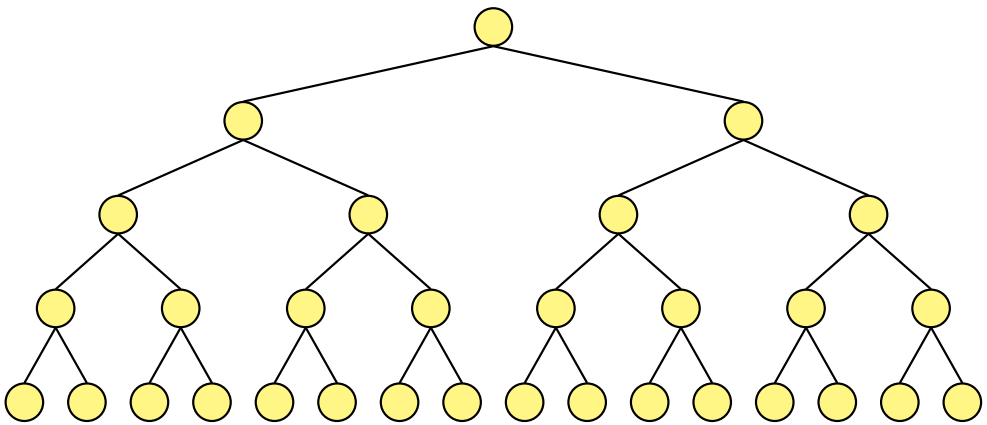
Ex: 31 nós

Eficiência da busca

Qual é o custo computacional da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$

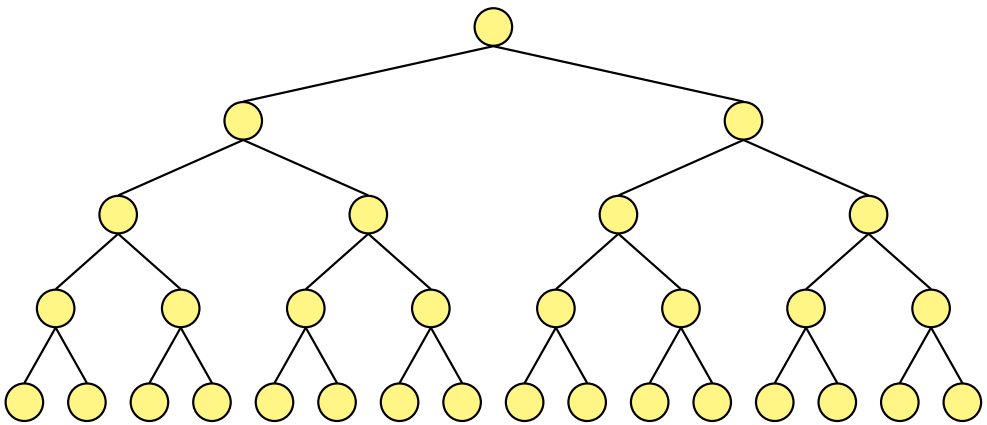
Árvore binária completa.

Eficiência da busca

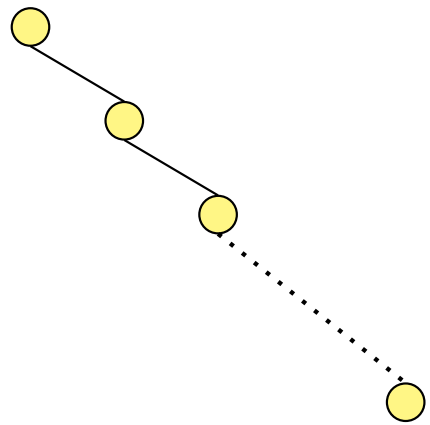
Qual é o custo computacional da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $\mathrm{O}(\lg n)$



Pior árvore: $\mathrm{O}(n)$

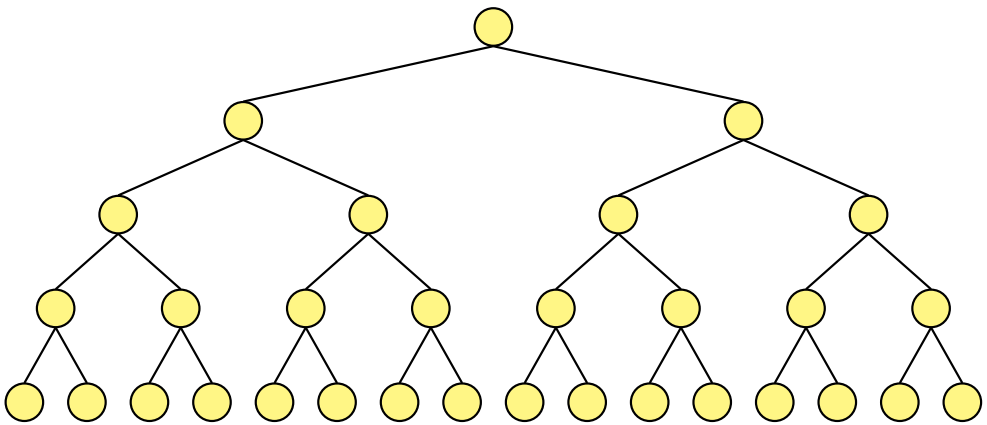
Para obter a pior árvore basta inserir em ordem crescente...

Eficiência da busca

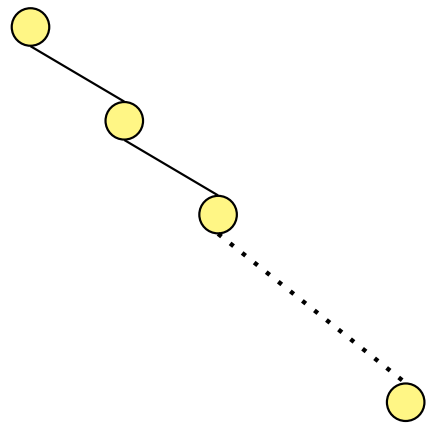
Qual é o custo computacional da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$



Pior árvore: $O(n)$

- Caso médio: em uma árvore com n elementos **adicionados aleatoriamente**, a busca demora (em média) $O(\lg n)$

Roteiro

ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

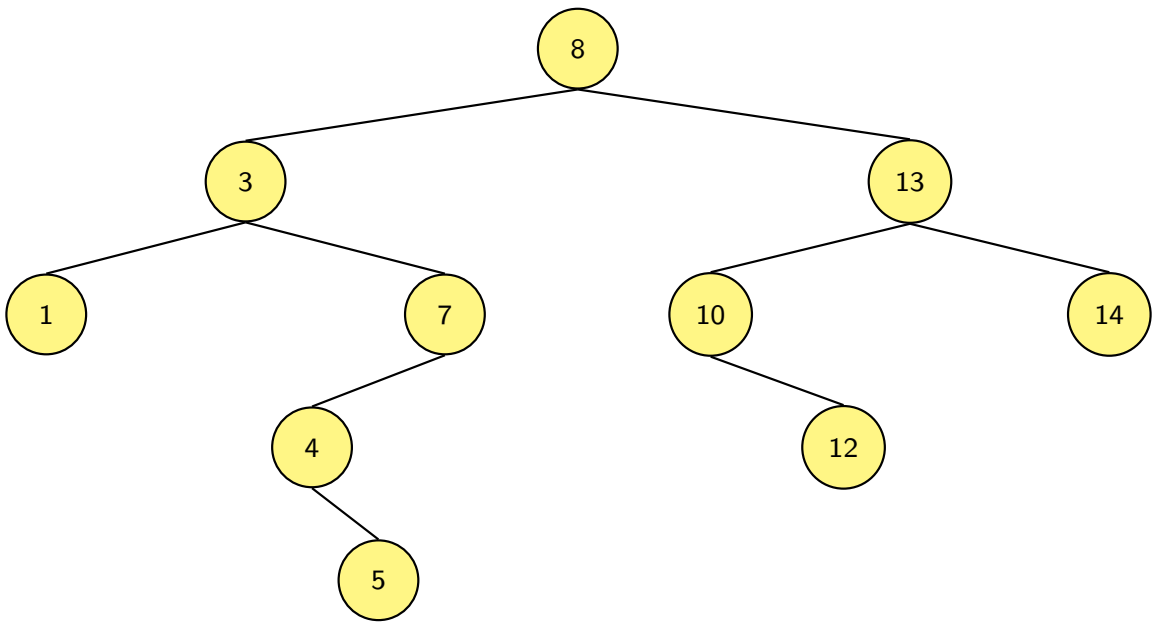


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

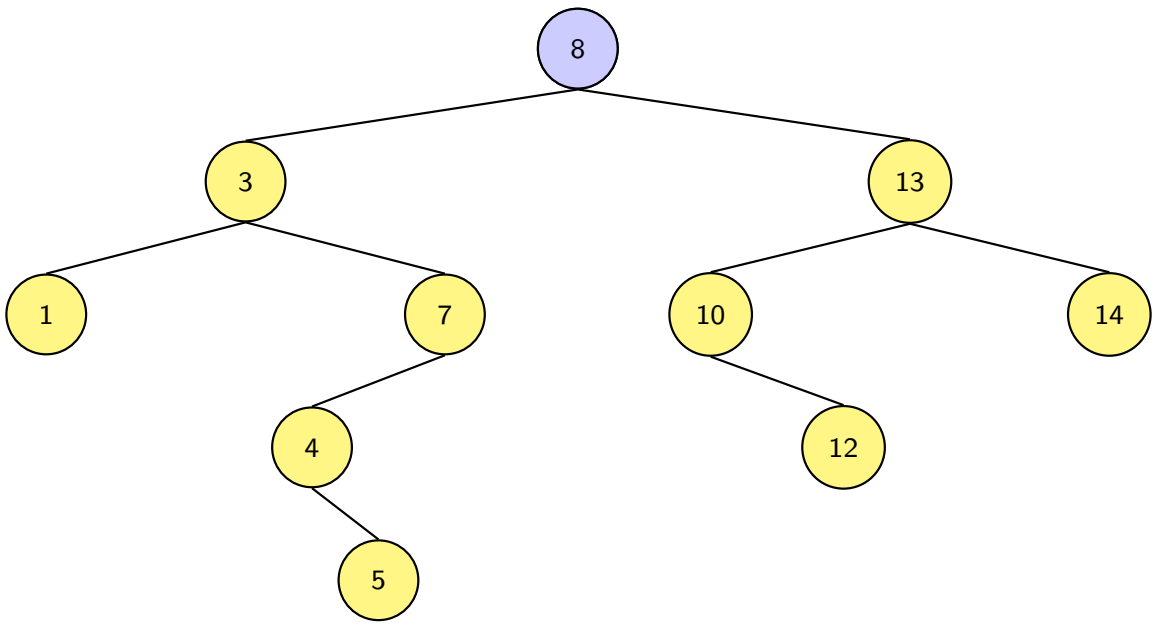


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

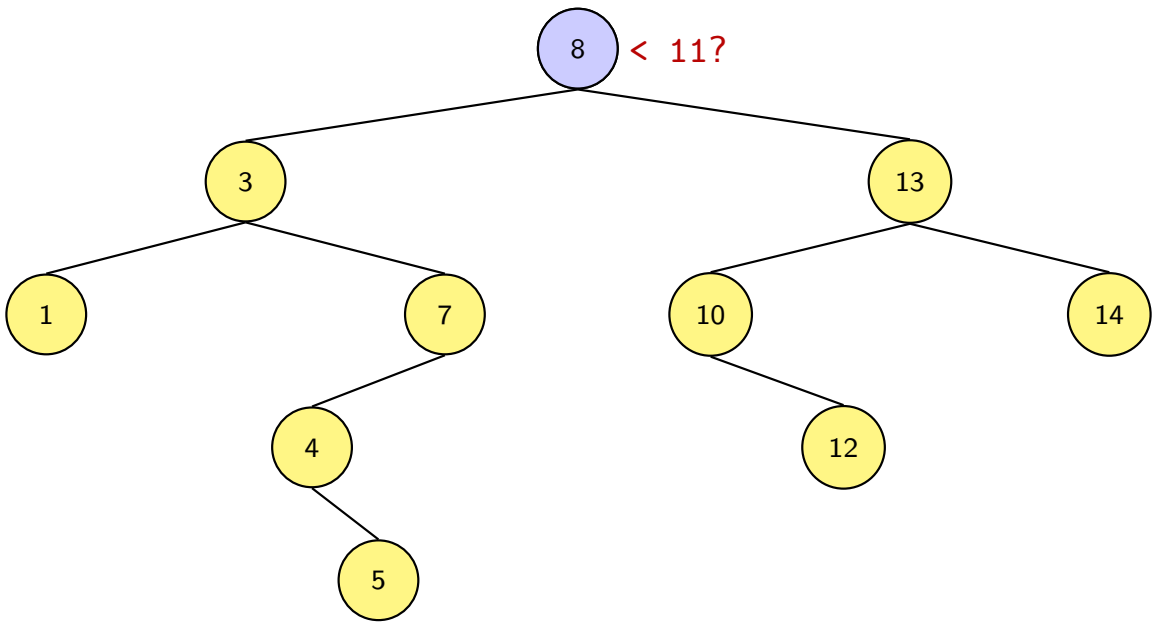


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

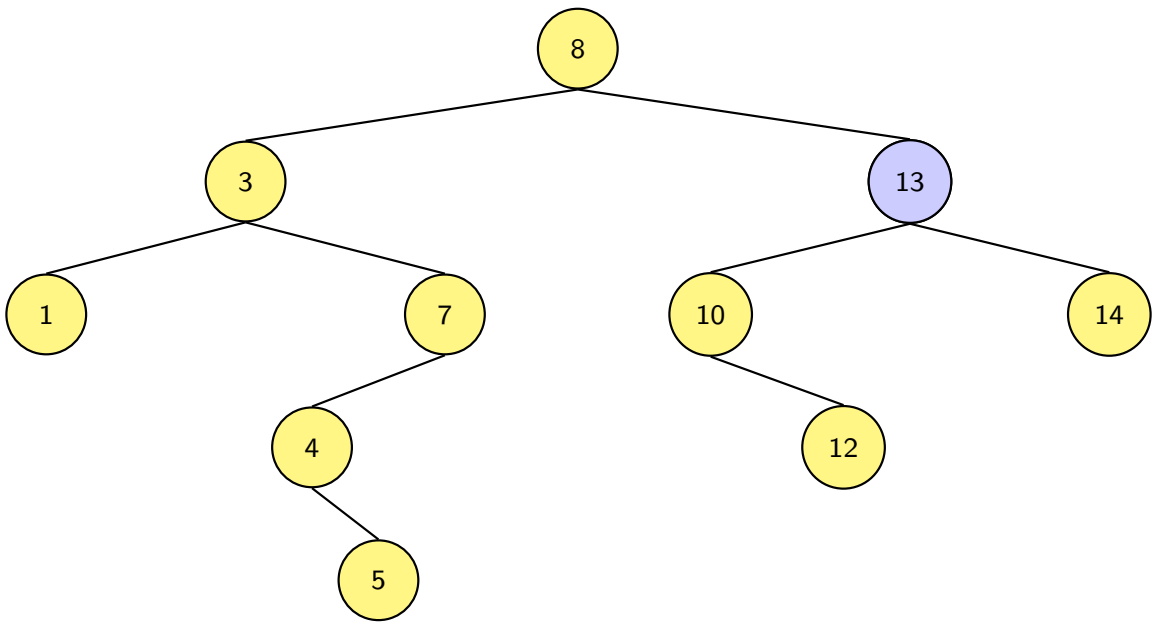


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

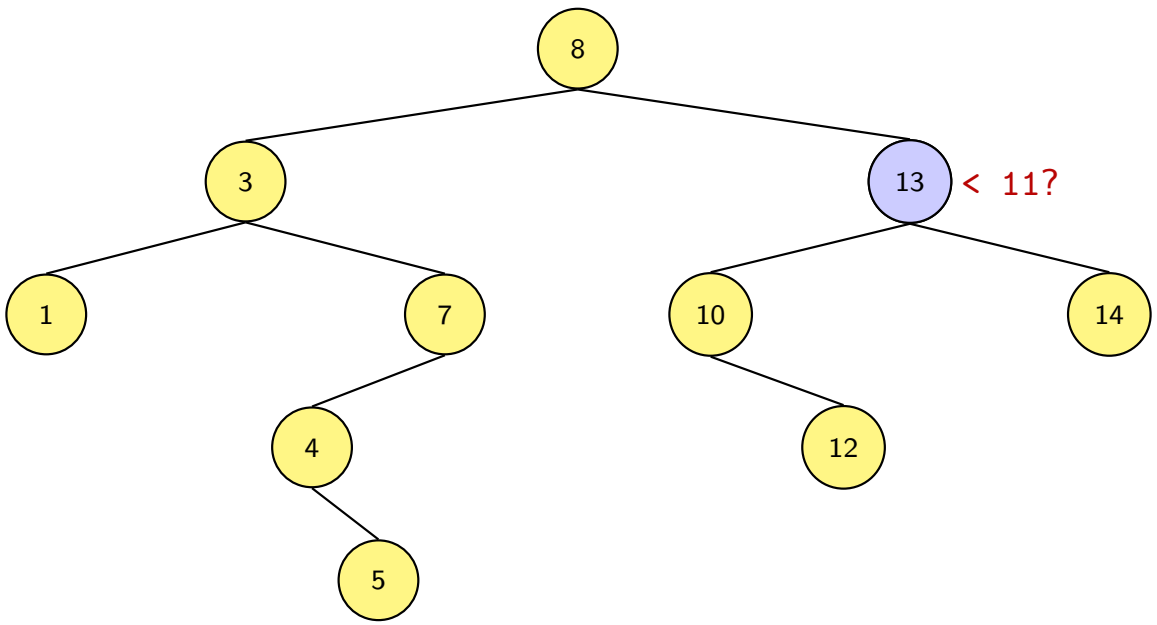


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

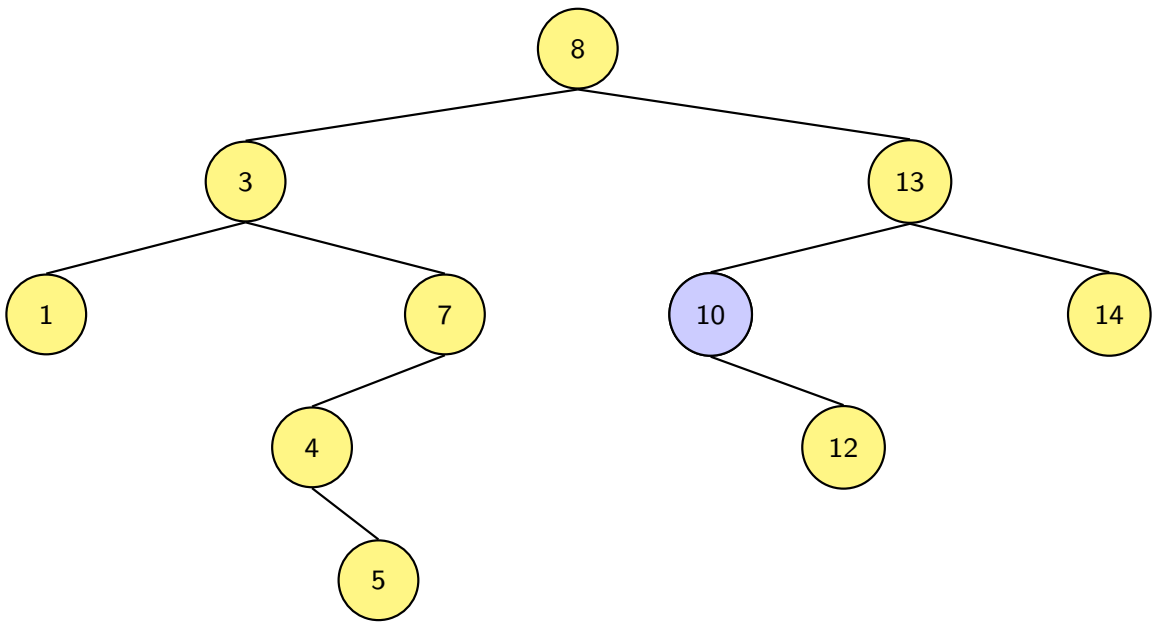


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

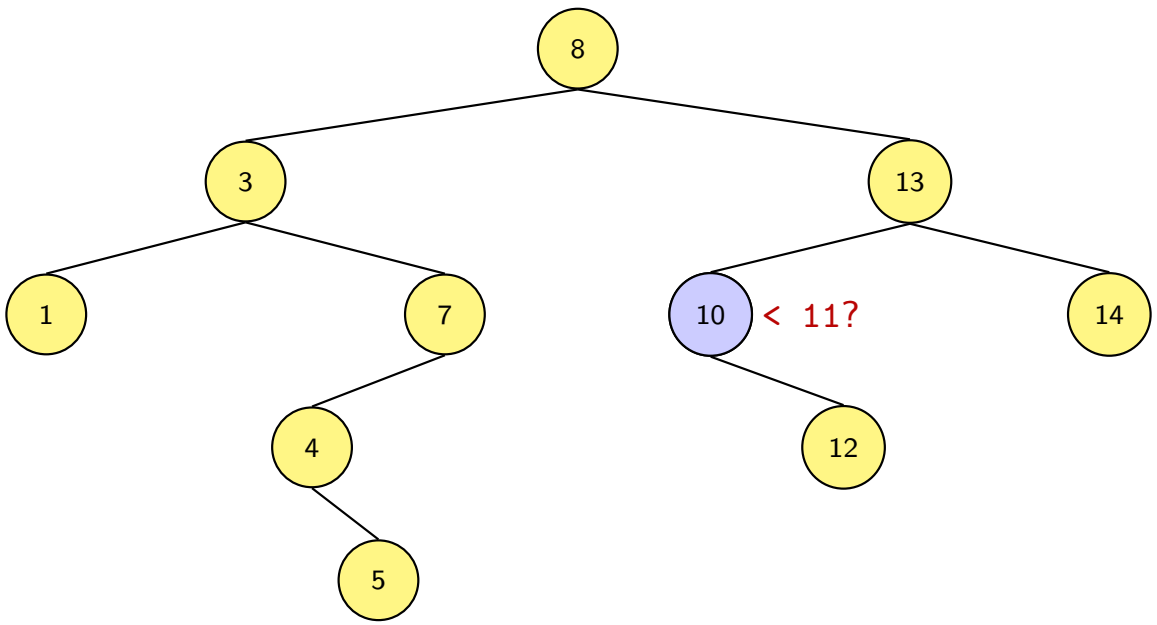


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

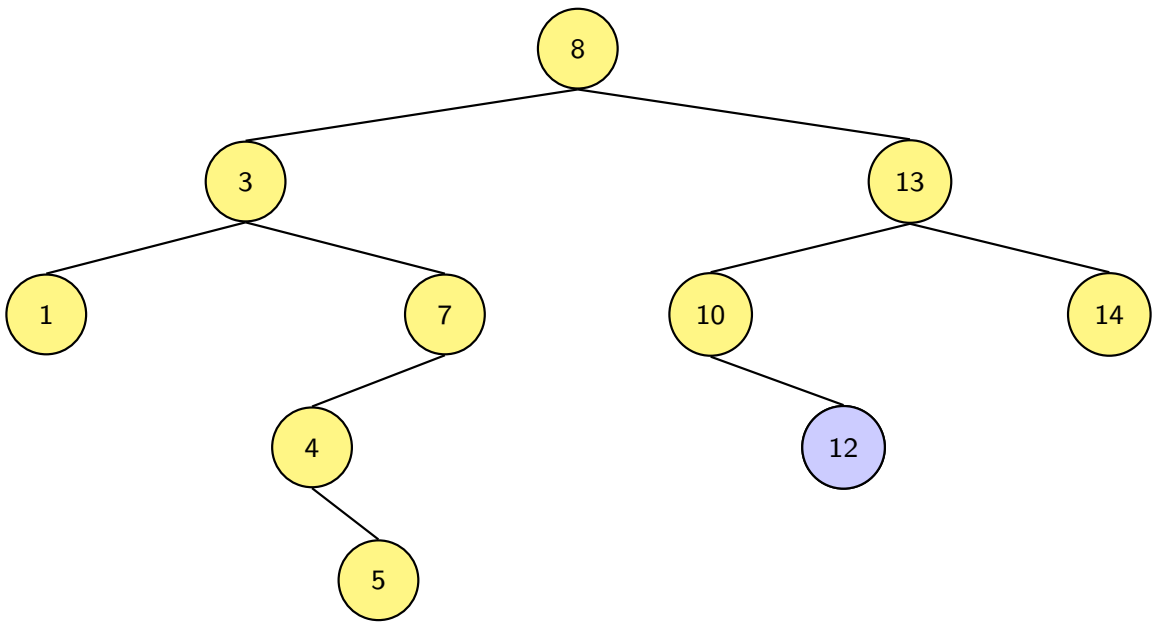


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

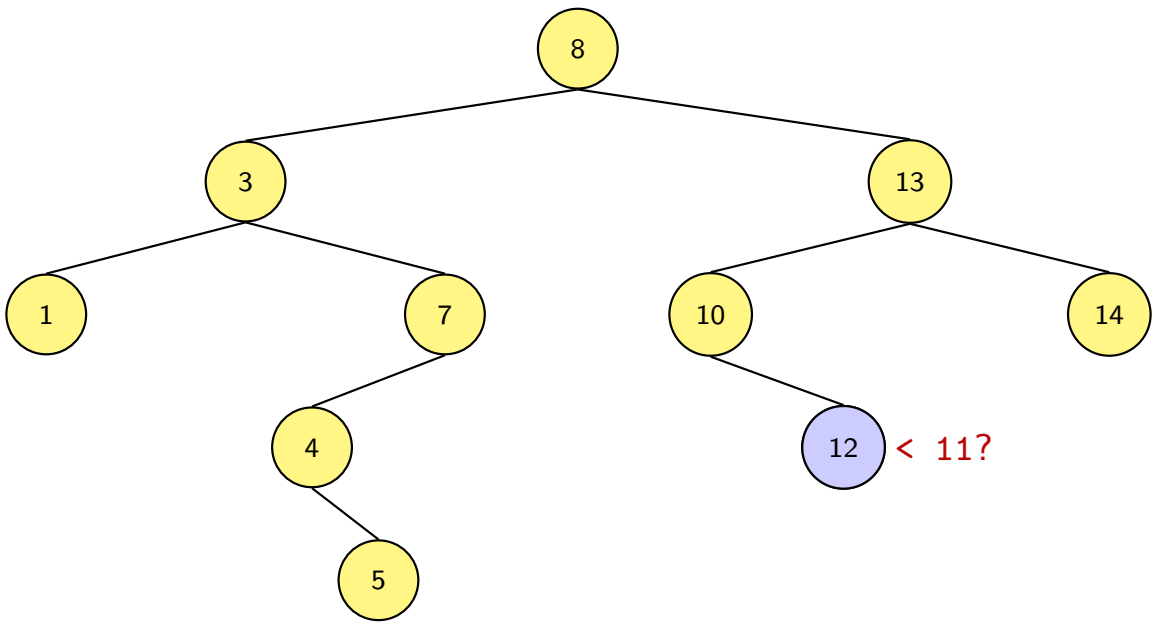


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

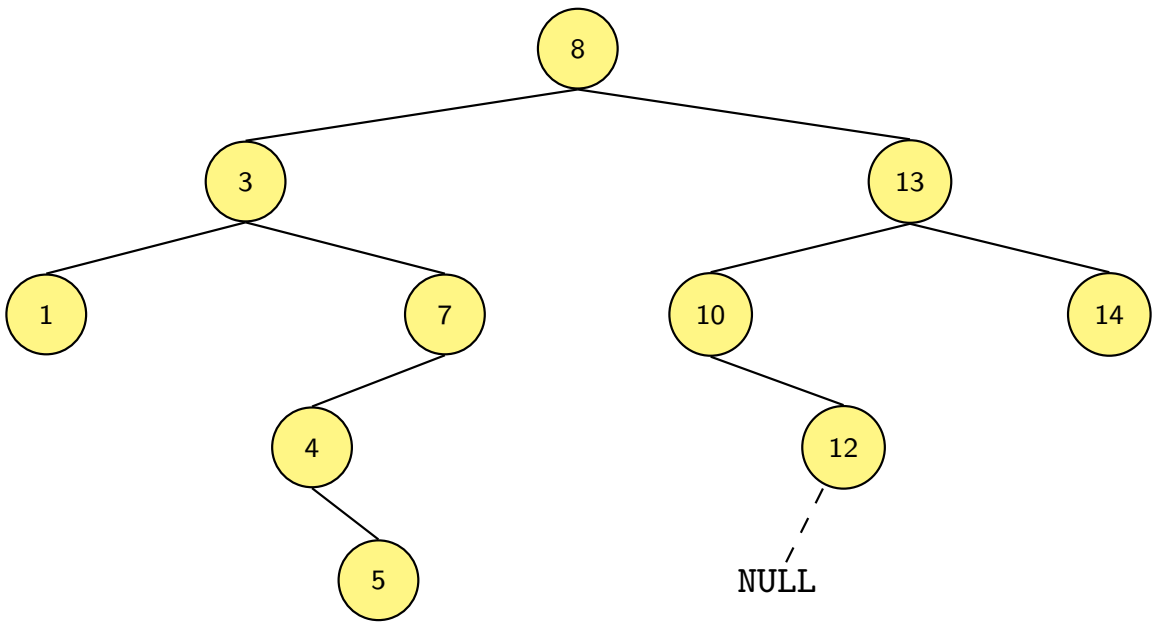


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

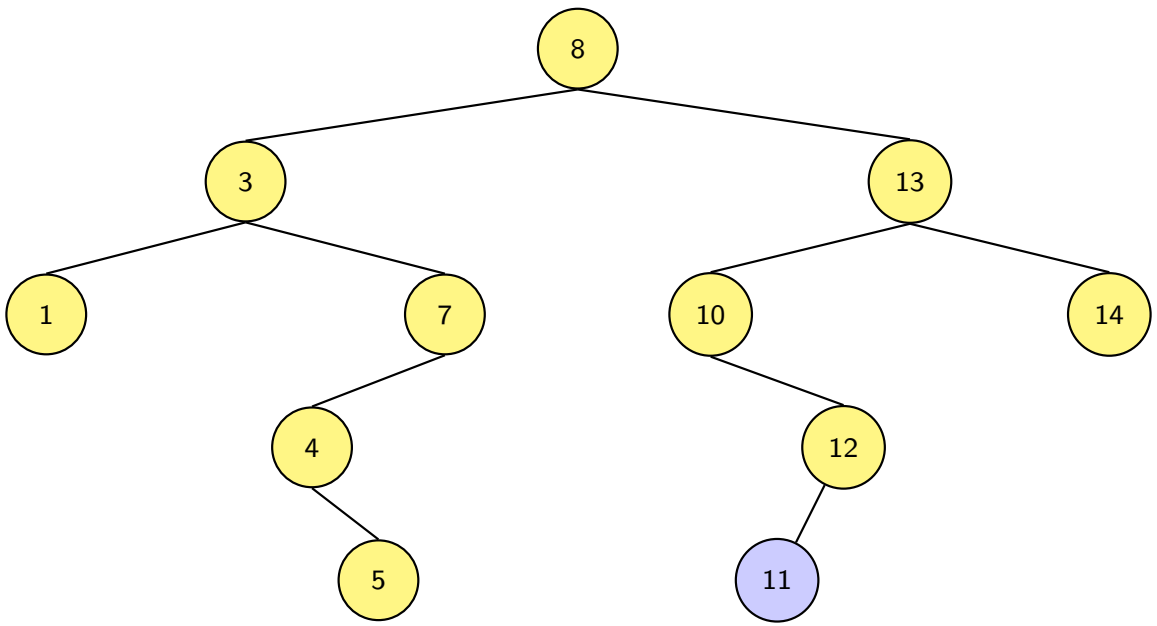
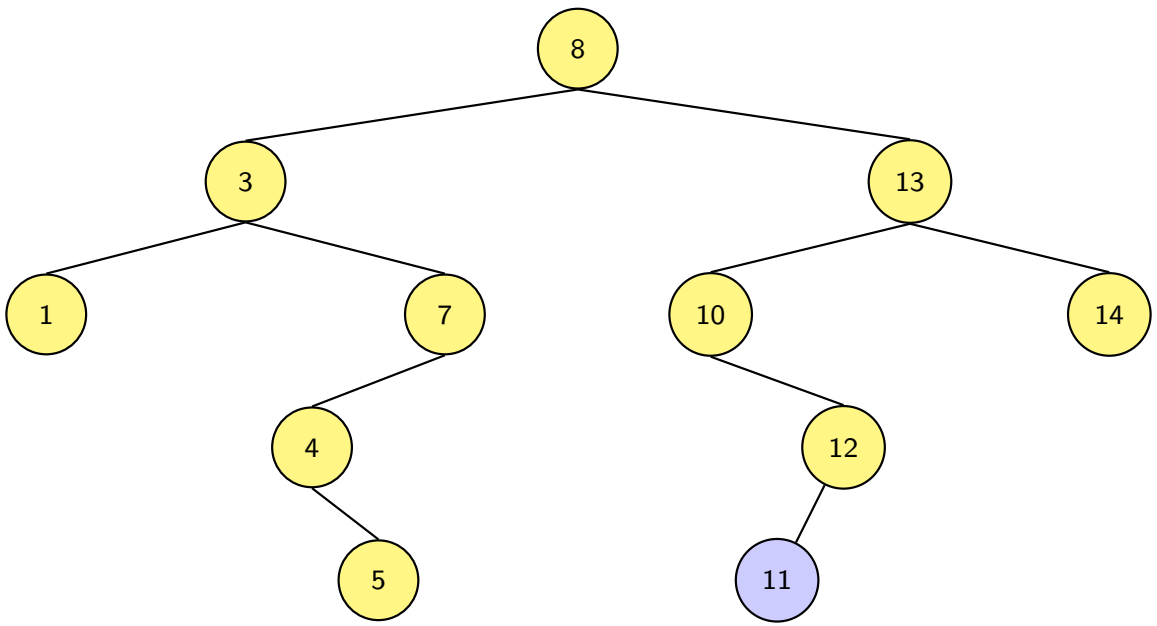


ABB - Inserir um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e inserimos onde ele **deveria estar**

Ex: Inserindo 11

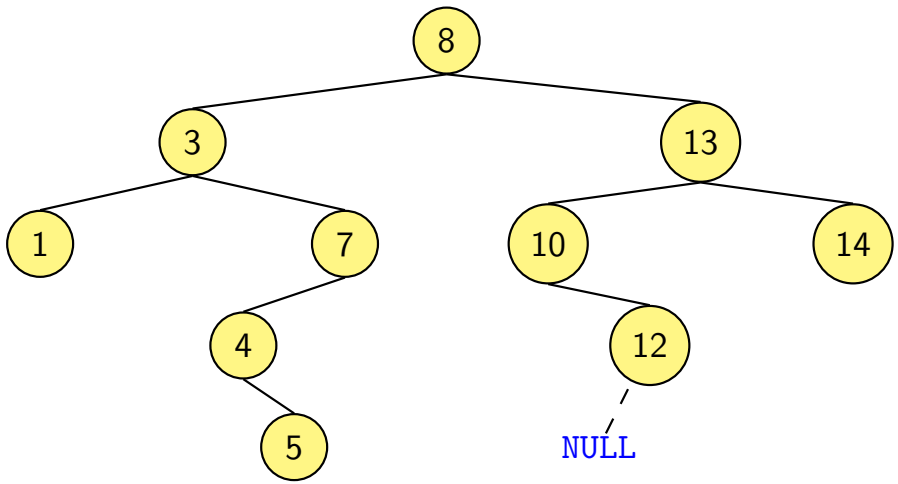


Esse algoritmo não garante uma árvore bem balanceada.

ABB - Inserir um valor

abb.c

```
54 No *inserir(No *p, int chave) {
55   No *novo;
56   if (p == NULL) {
57     novo = malloc(sizeof(No));
58     novo->esq = novo->dir = NULL;
59     novo->chave = chave;
60     return novo;
61   }
62   if (chave < p->chave)
63     p->esq = inserir(p->esq, chave);
64   else
65     p->dir = inserir(p->dir, chave);
66   return p;
67 }
```



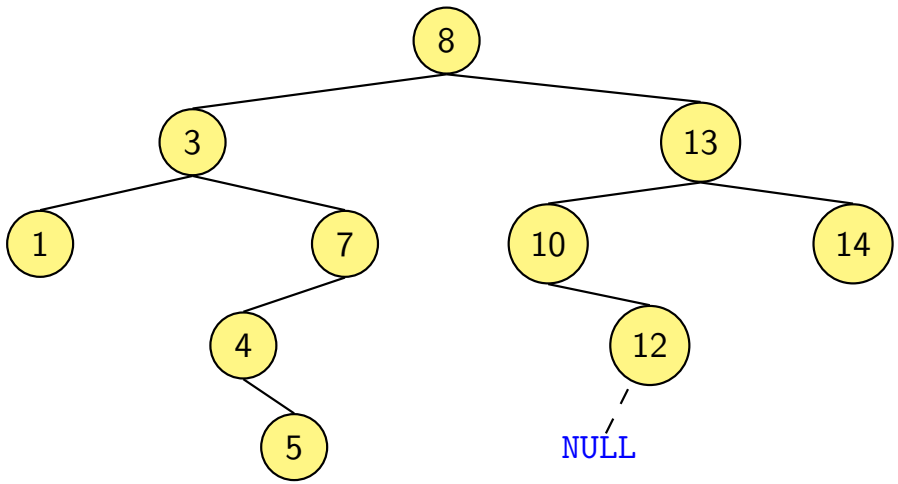
O algoritmo insere na árvore recursivamente:

- devolve um ponteiro para a raiz da "nova" árvore

ABB - Inserir um valor

abb.c

```
54 No *inserir(No *p, int chave) {
55   No *novo;
56   if (p == NULL) {
57     novo = malloc(sizeof(No));
58     novo->esq = novo->dir = NULL;
59     novo->chave = chave;
60     return novo;
61   }
62   if (chave < p->chave)
63     p->esq = inserir(p->esq, chave);
64   else
65     p->dir = inserir(p->dir, chave);
66   return p;
67 }
```



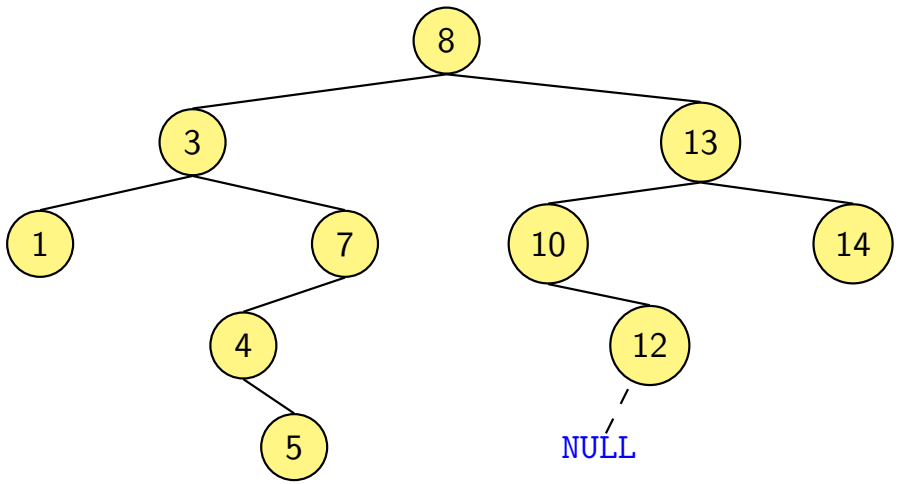
Custo computacional[1]:

[1] Depende da altura da árvore.

ABB - Inserir um valor

abb.c

```
54 No *inserir(No *p, int chave) {
55   No *novo;
56   if (p == NULL) {
57     novo = malloc(sizeof(No));
58     novo->esq = novo->dir = NULL;
59     novo->chave = chave;
60     return novo;
61   }
62   if (chave < p->chave)
63     p->esq = inserir(p->esq, chave);
64   else
65     p->dir = inserir(p->dir, chave);
66   return p;
67 }
```



Custo computacional[1]:

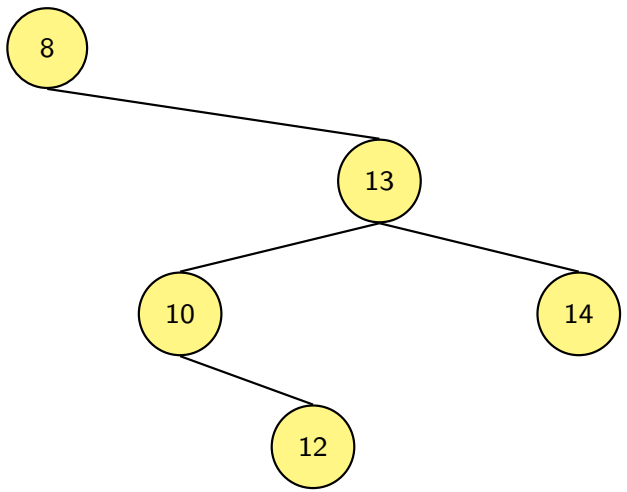
- No pior caso $1 + 2 + 3 + ... + n = O(n^2)$

[1] Depende da altura da árvore.

Roteiro

Mínimo da Árvore

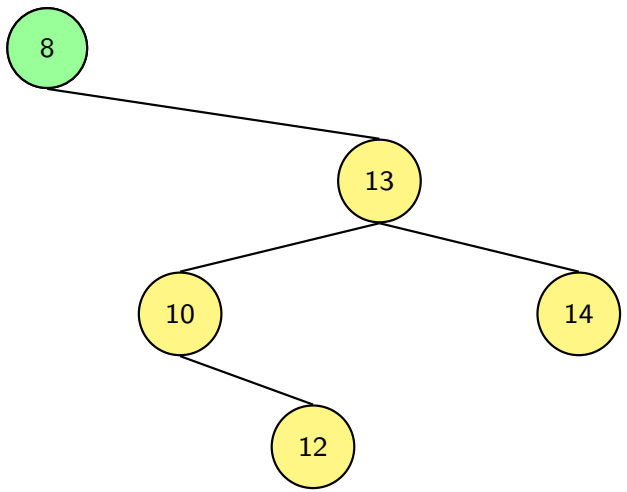
Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

Mínimo da Árvore

Onde está o nó com a menor chave de uma árvore?

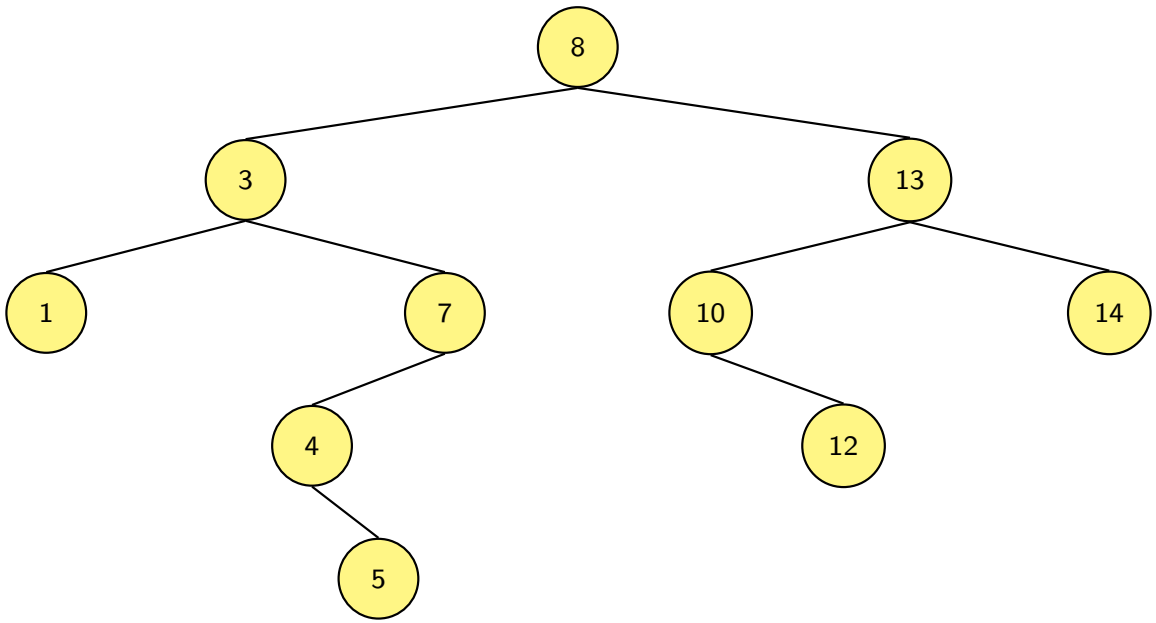


Quem é o mínimo para essa árvore?

- É a própria raiz

Mínimo da Árvore

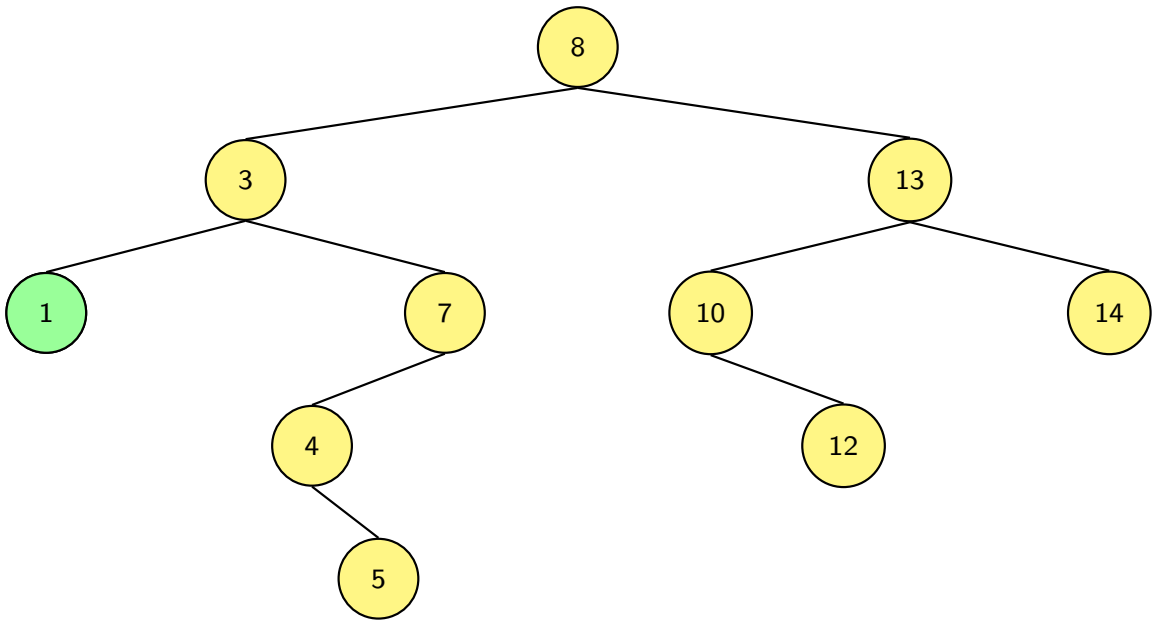
Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

Mínimo da Árvore

Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

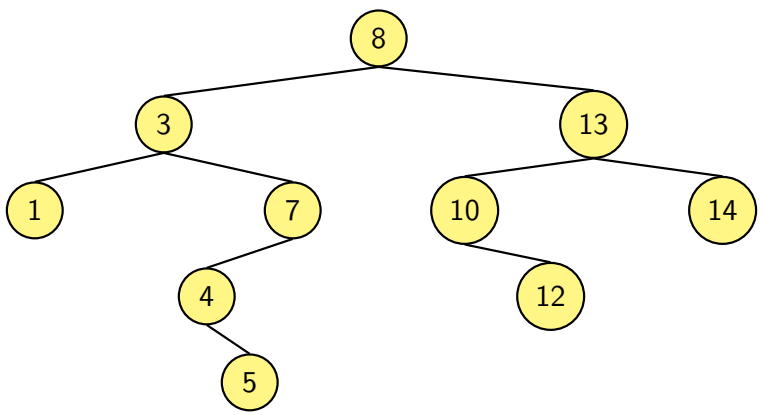
- É o mínimo da **subárvore esquerda**

Mínimo - Implementações

abb.c

```
107 No* minimo(No *p) {//recursão em cauda
108   if (p == NULL || p->esq == NULL)
109     return p;
110   return minimo(p->esq);
111 }
```

```
113 No* minimo_iterativo(No *p) {
114   while (p != NULL && p->esq != NULL)
115     p = p->esq;
116   return p;
117 }
```

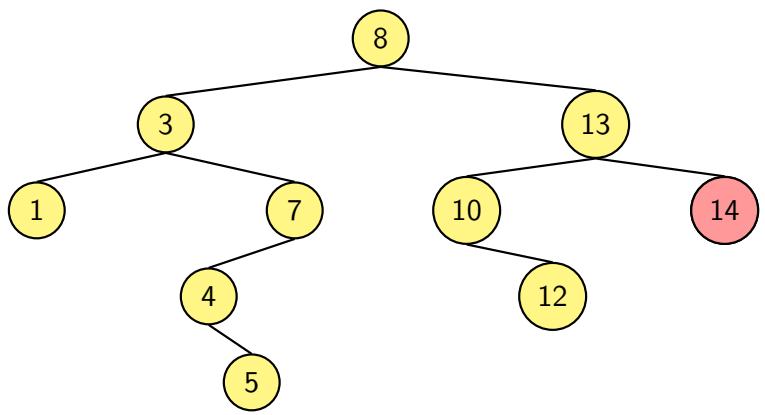


Mínimo - Implementações

abb.c

```
107 No* minimo(No *p) {//recursão em cauda
108   if (p == NULL || p->esq == NULL)
109     return p;
110   return minimo(p->esq);
111 }
```

```
113 No* minimo_iterativo(No *p) {
114   while (p != NULL && p->esq != NULL)
115     p = p->esq;
116   return p;
117 }
```

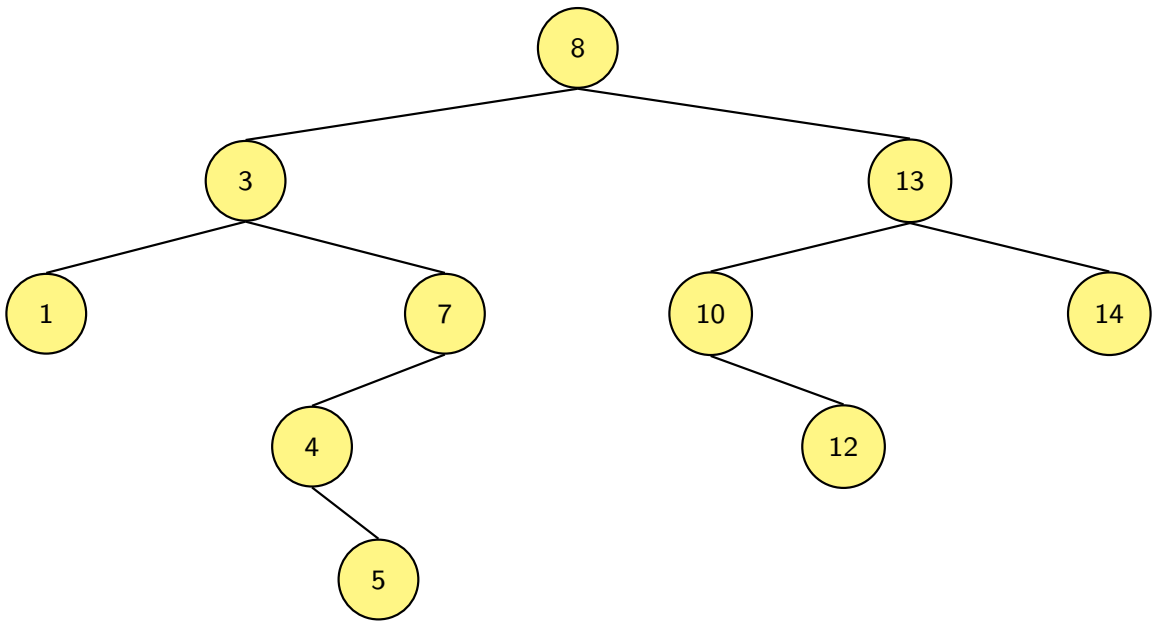


Para encontrar o máximo, basta fazer a operação simétrica

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

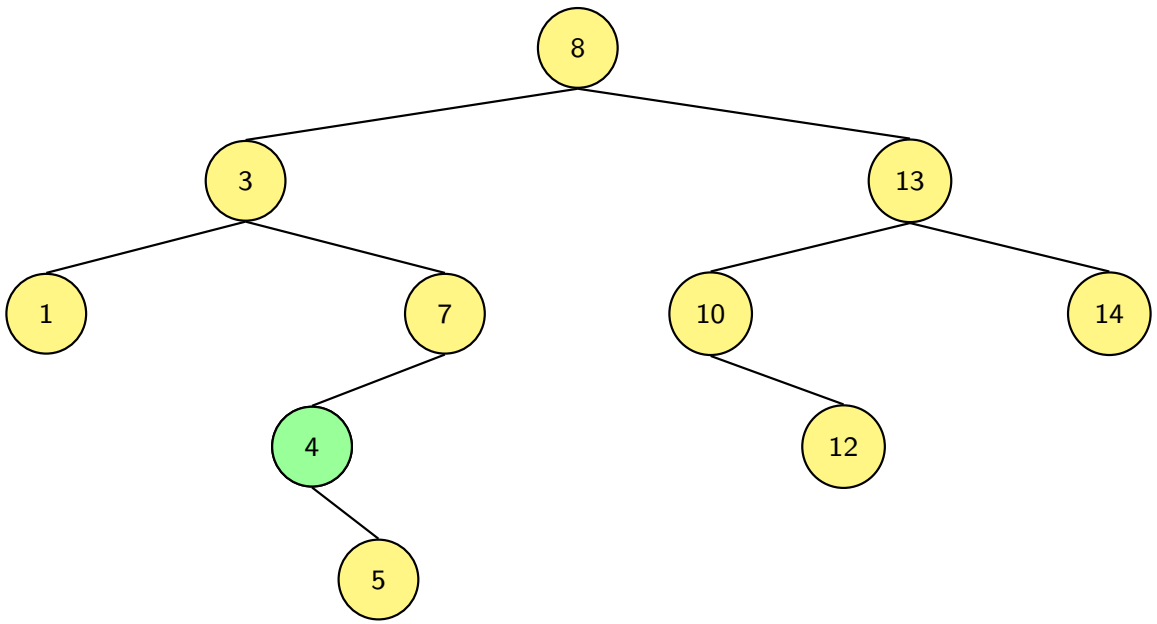


Quem é o sucessor de 3?

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



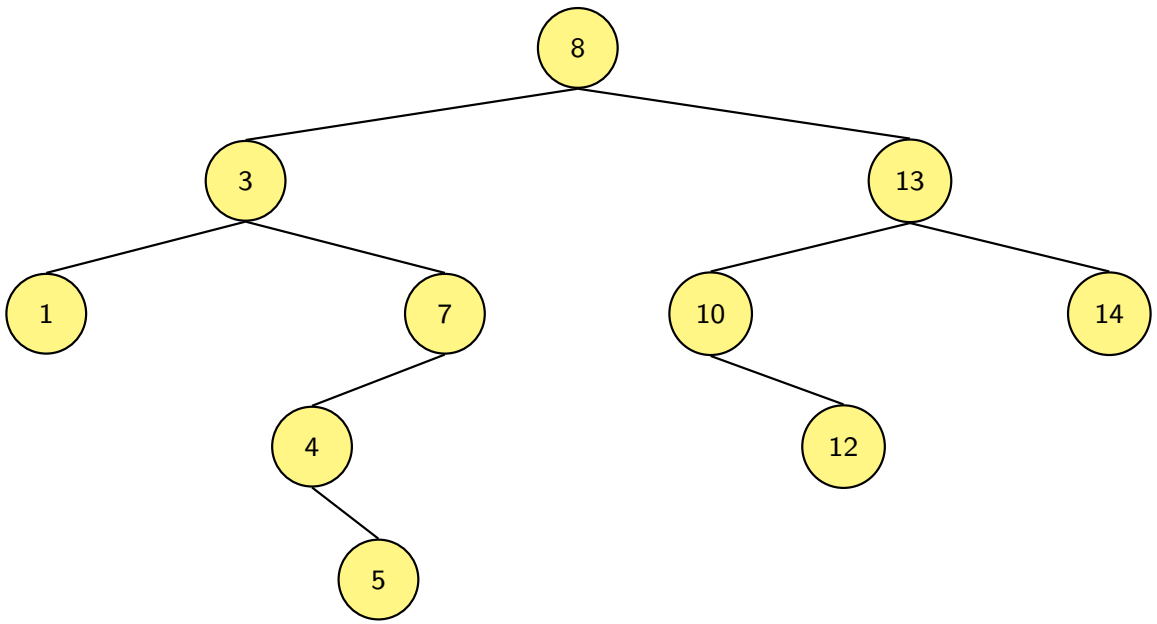
Quem é o sucessor de 3?

- É o mínimo da sua subárvore direita de 3

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

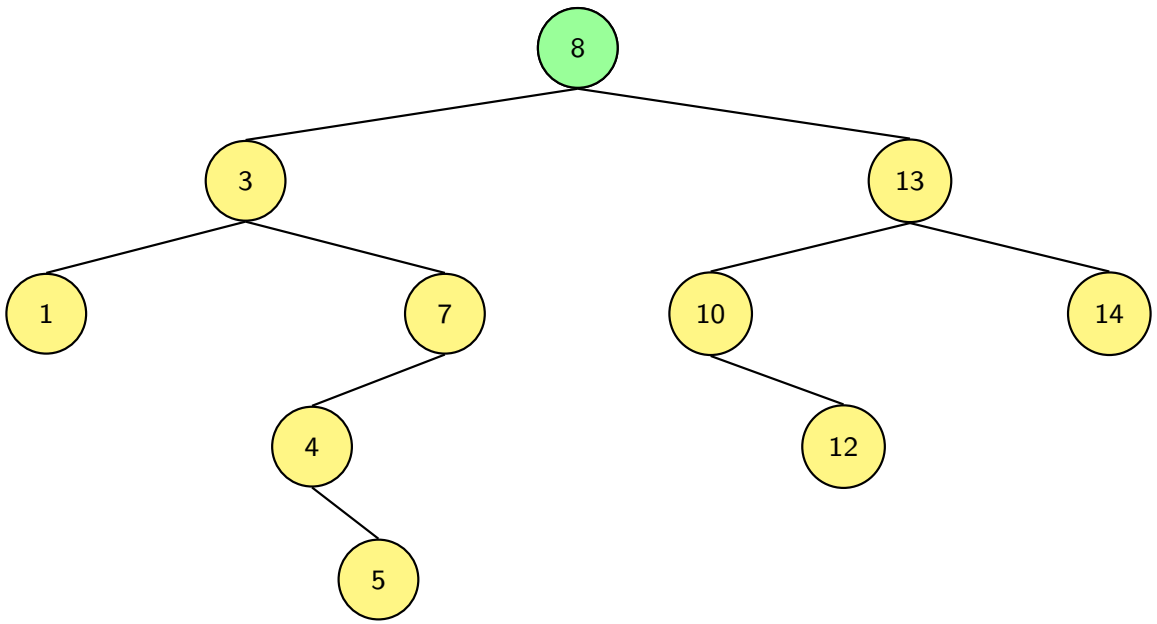


Quem é o sucessor de 7?

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



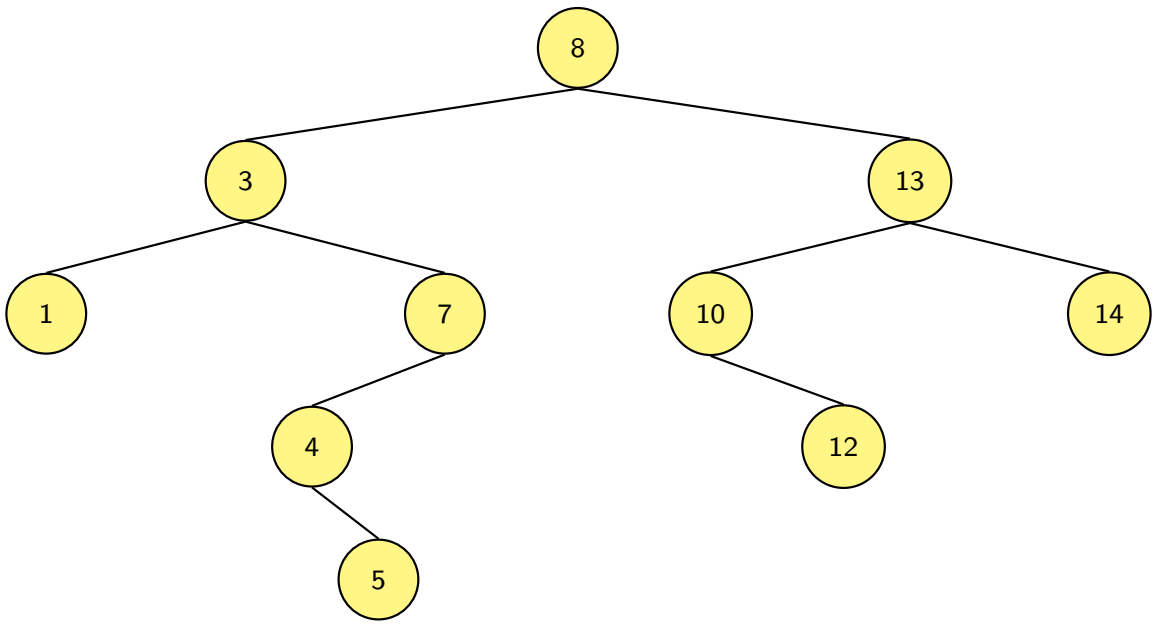
Quem é o sucessor de 7?

- É primeiro ancestral a direita

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

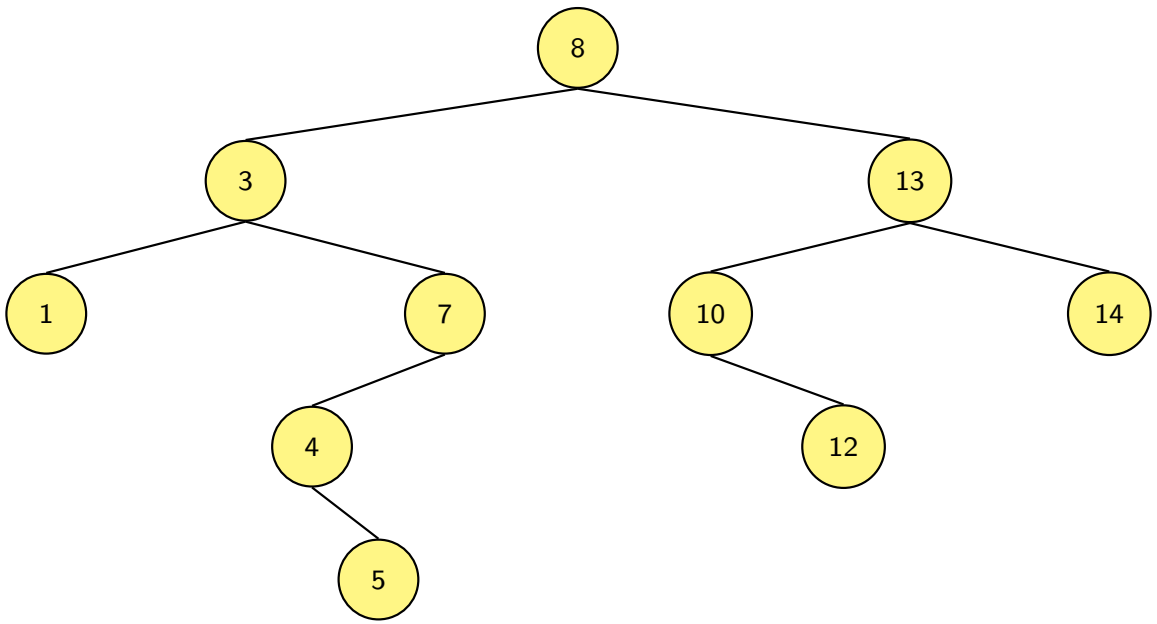


Quem é o sucessor de 14?

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



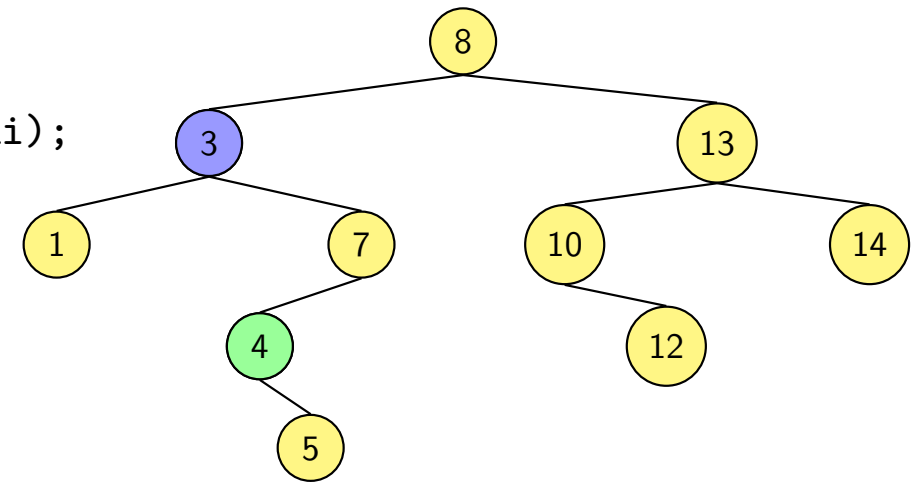
Quem é o sucessor de 14?

- não tem sucessor...

Sucessor - Implementação

abb.c

```
119 No* ancestral_a_direita(No *x) {
120   if (x == NULL)
121     return NULL;
122   if (x->pai == NULL || x->pai->esq == x)
123     return x->pai;
124   else
125     return ancestral_a_direita(x->pai);
126 }
127
128 No* sucessor(No *x) {
129   if (x->dir != NULL)
130     return minimo(x->dir);
131   else
132     return ancestral_a_direita(x);
133 }
```



A implementação da função `antecessor` é simétrica.

ABB - Remover um valor

Ex: removendo 10

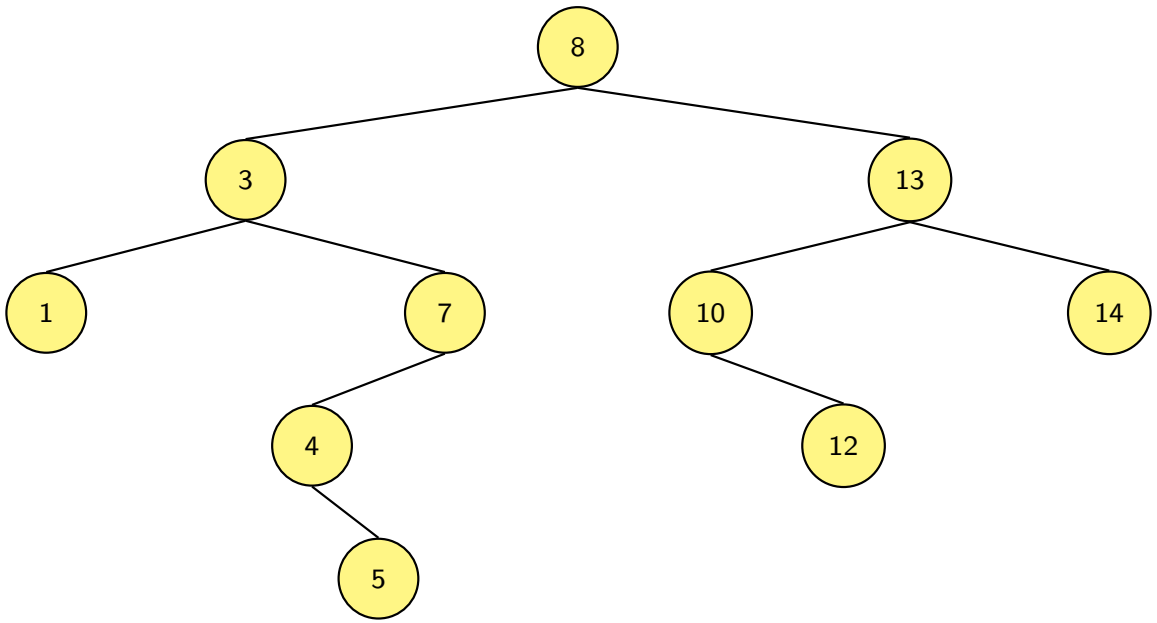


ABB - Remover um valor

Ex: removendo 10

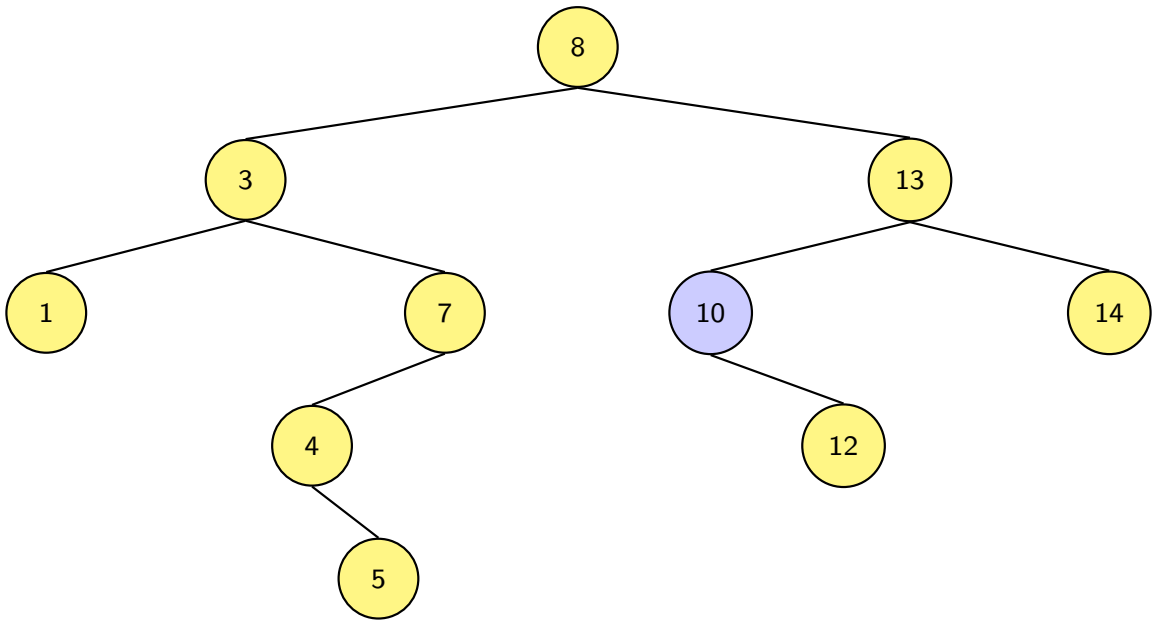


ABB - Remover um valor

Ex: removendo 10

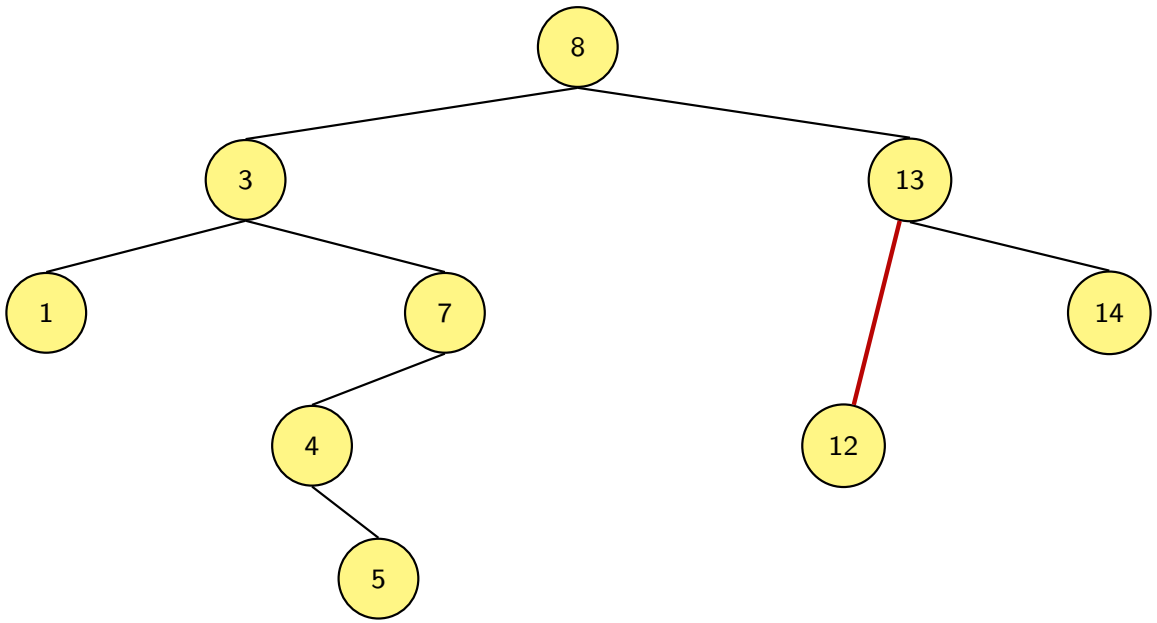


ABB - Remover um valor

Ex: removendo 3

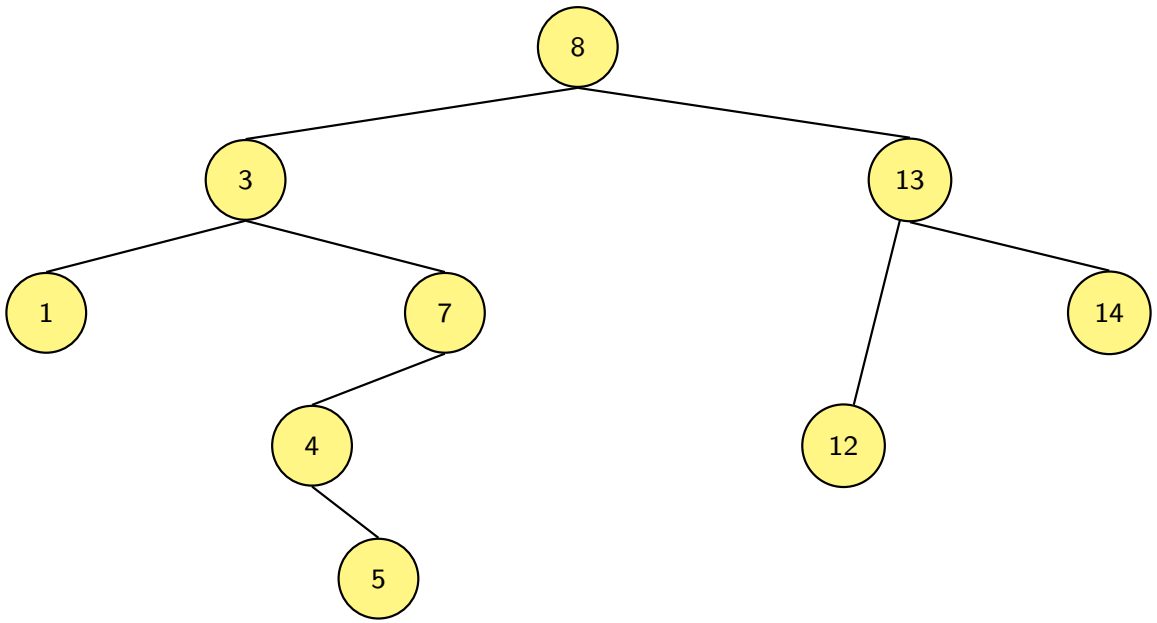


ABB - Remover um valor

Ex: removendo 3

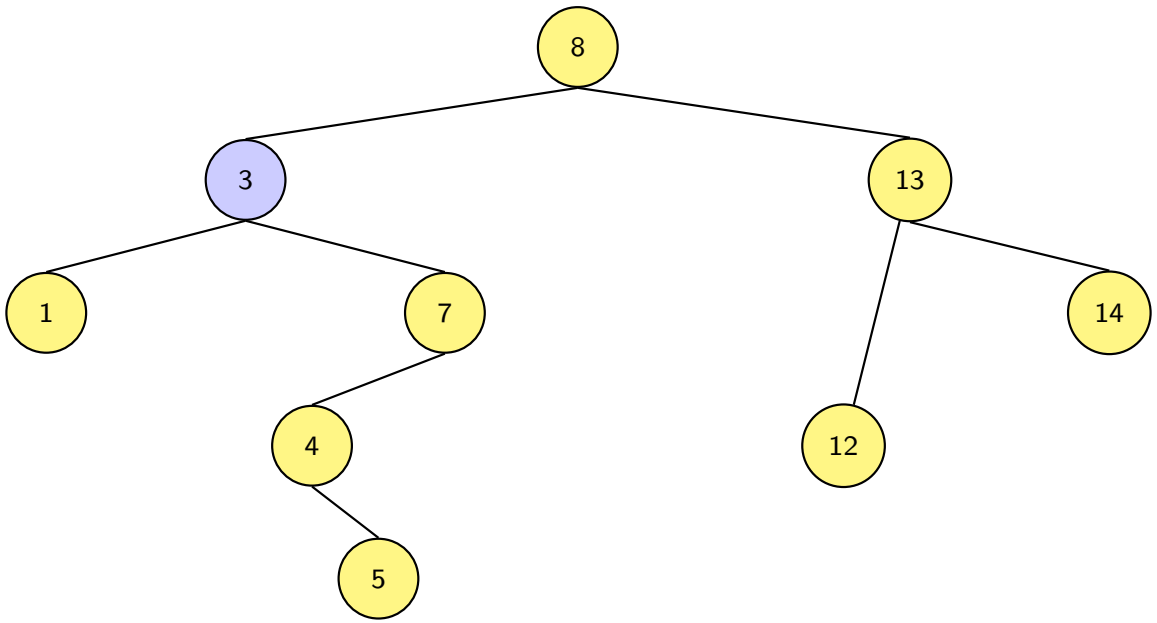
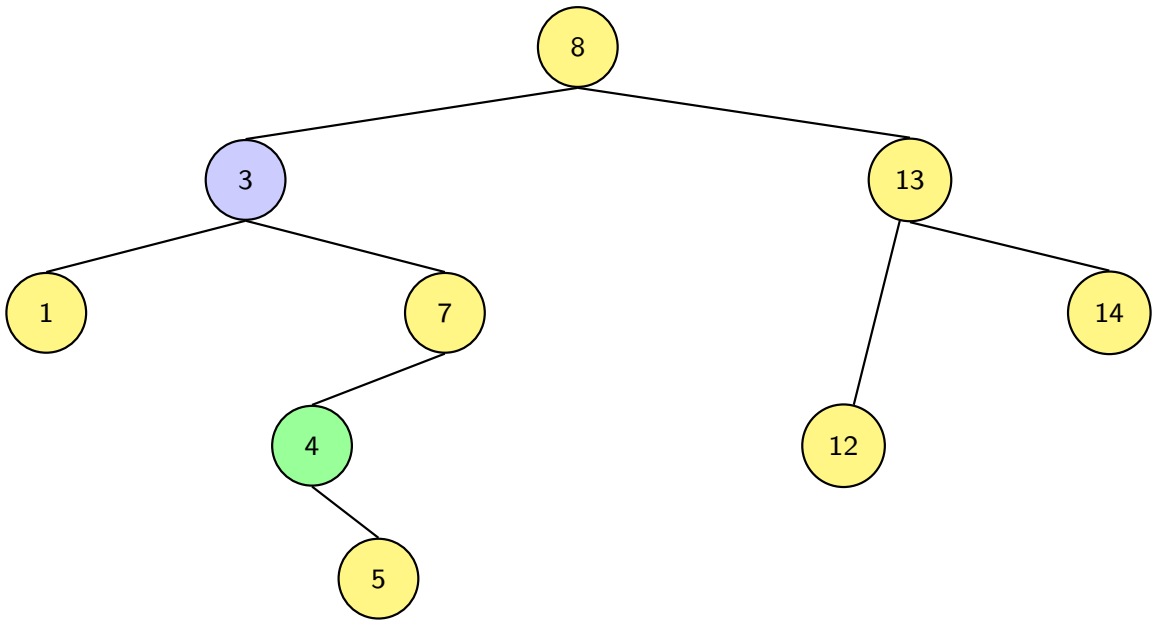


ABB - Remover um valor

Ex: removendo 3

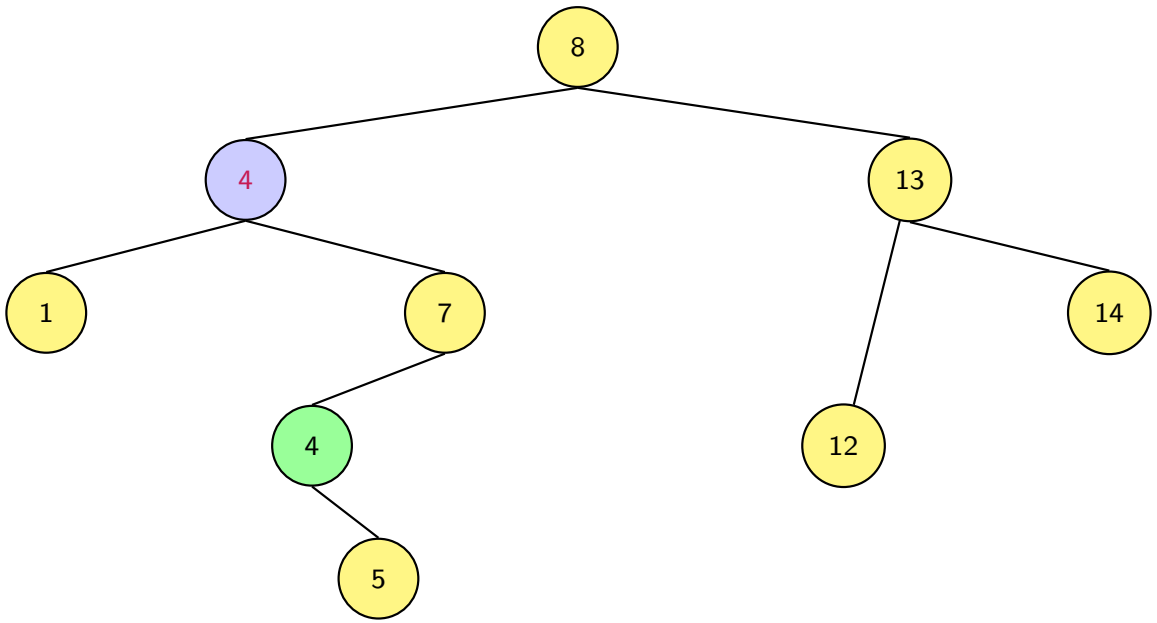


Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da **árvore binária de busca**

ABB - Remover um valor

Ex: removendo 3



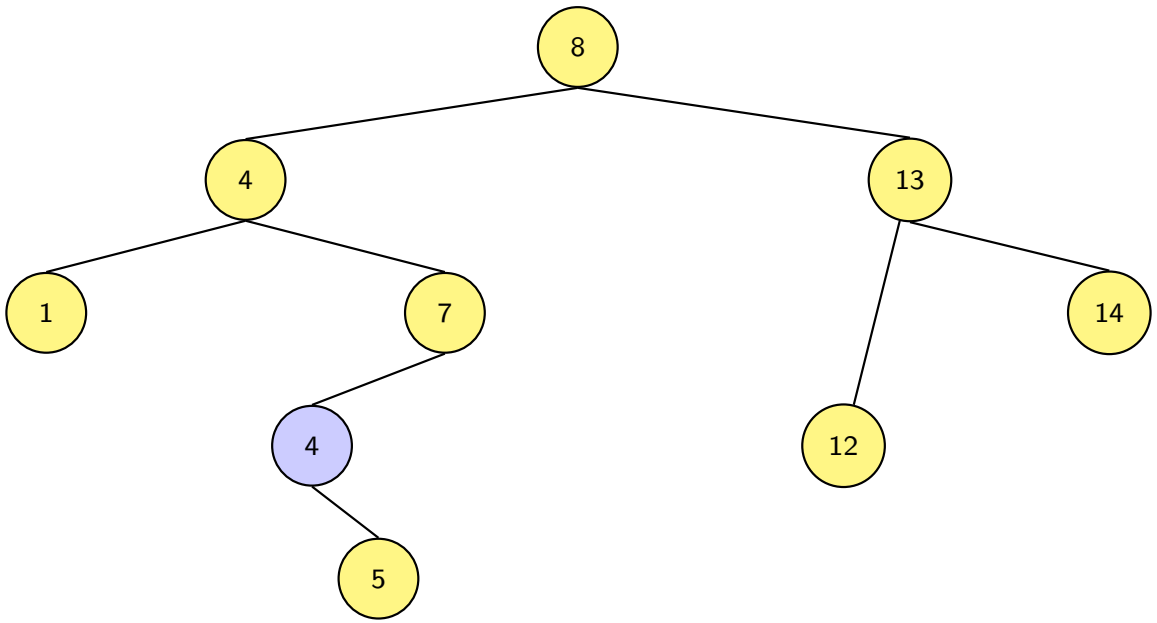
Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da **árvore binária de busca**

E agora removemos o sucessor

ABB - Remover um valor

Ex: removendo 3



Podemos colocar o sucessor de 3 em seu lugar

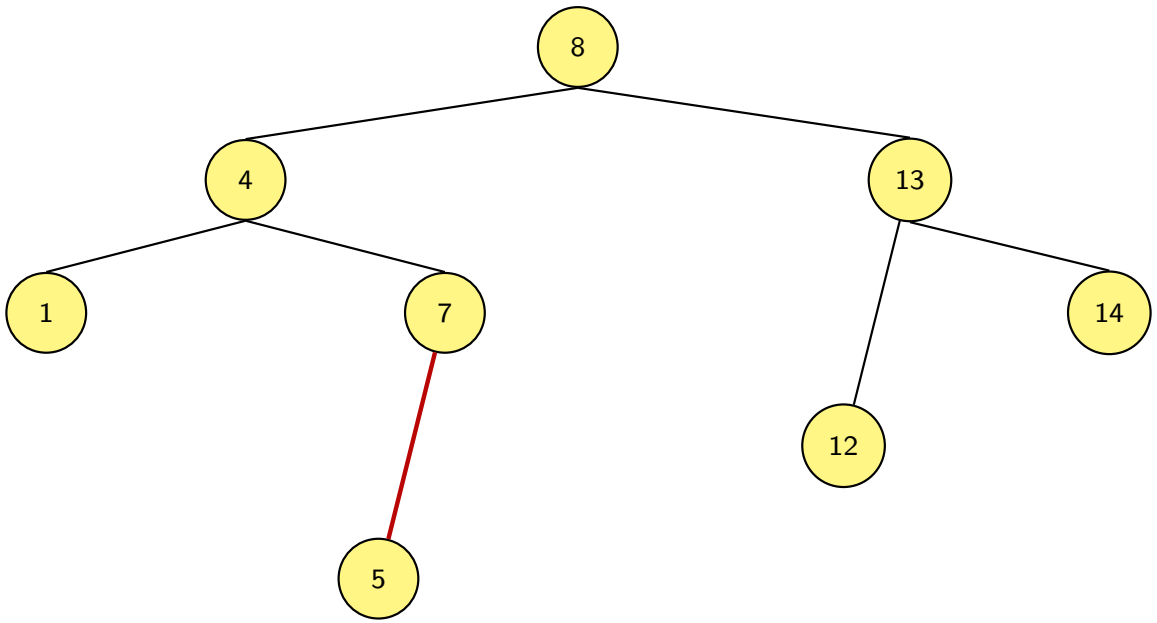
- Isso mantém a propriedade da **árvore binária de busca**

E agora removemos o sucessor

- O sucessor nunca tem **filho esquerdo**!

ABB - Remover um valor

Ex: removendo 3



Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da **árvore binária de busca**

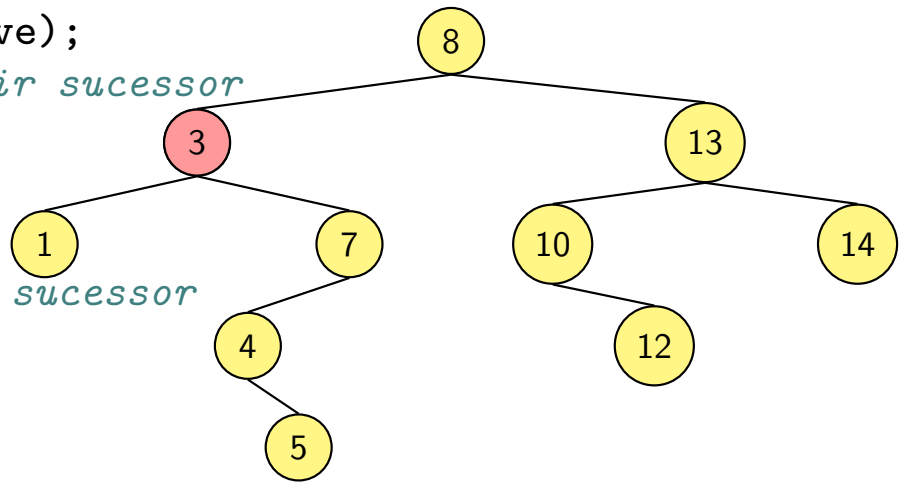
E agora removemos o sucessor

- O sucessor nunca tem **filho esquerdo**!

ABB - Remover um valor

abb.c

```
83  No* remover_rec(No *p, int chave) {
84    if (p == NULL) return NULL;
85    if (chave < p->chave)
86      p->esq = remover_rec(p->esq, chave);
87    else if (chave > p->chave)
88      p->dir = remover_rec(p->dir, chave);
89    else if (p->esq == NULL){//filho dir sucessor
90      No *q = p->dir; free(p);
91      return q;
92    }
93    else if (p->dir == NULL){//não tem sucessor
94      No *q = p->esq; free(p);
95      return q;
96    }
97    else remover_sucessor(p);
98    return p;
99  }
```

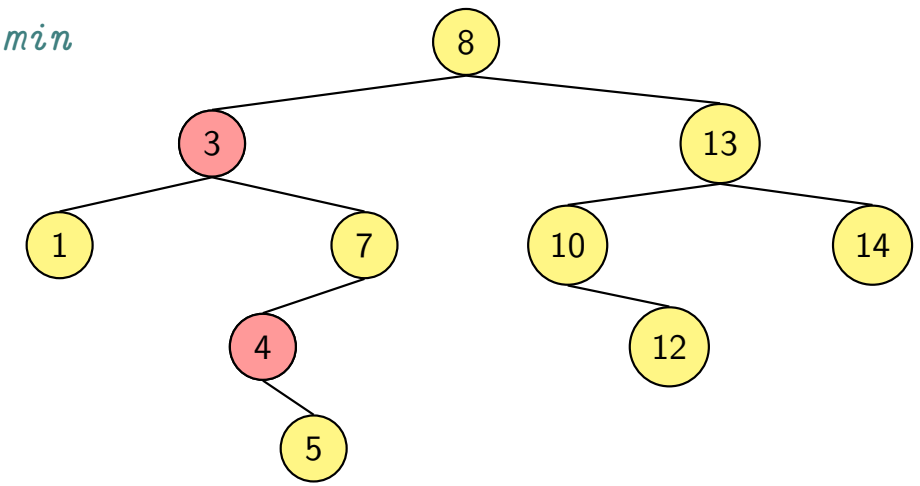



```
101 void remover(No **p, int chave) {
102   *p = remover_rec(*p, chave);
103 }
```

ABB - Remover um valor

abb.c

```
69 void remover_sucessor(No *p) {
70   No *min = p->dir; //será o mínimo da subárvore direita
71   No *pai = p;      //será o pai de min
72   while (min->esq != NULL) {
73     pai = min;
74     min = min->esq;
75   }
76   if (pai->esq == min)
77     pai->esq = min->dir;
78   else
79     pai->dir = min->dir;
80   p->chave = min->chave; free(min);
81 }
```



Cliente

exemplo2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "abb.h"

int main() {
  int i, v[10] = {8, 3, 1, 7, 13, 10, 14, 12, 4, 5};
  No *T = criar_arvore();
  for (i = 0; i < 10; i++) T = inserir(T, v[i]);
  imprimir_arvore(T, 0);
  int chave = 0;
  while(chave!=-1){
    printf("> "); scanf("%d", &chave);
    if(buscar(T, chave)!=NULL){
      printf("REMOVIDO!!\n"); remover(&T, chave);
      imprimir_arvore(T, 0);
    }
    else printf("NAO\n");
  }
  destruir_arvore(&T);
return 0;
}
```

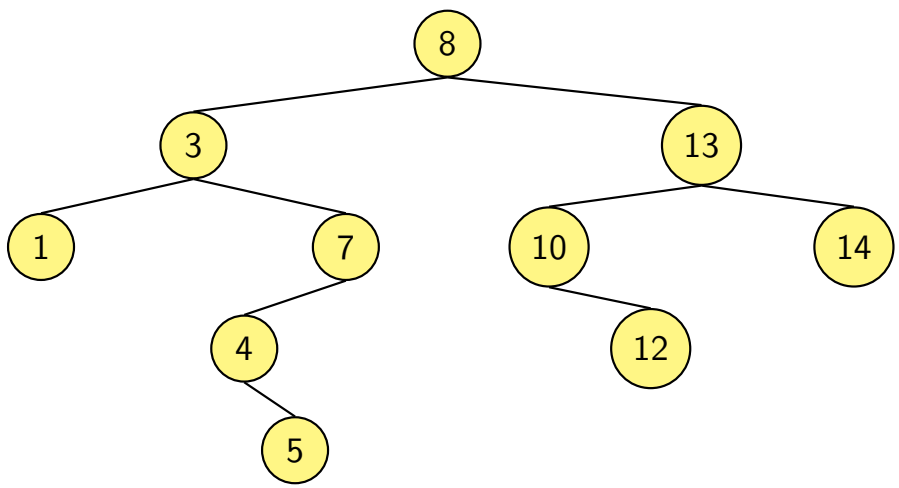
Makefile

Vamos usar o Makefile para compilar:

```
1 exemplo2: exemplo2.c abb.o
2   gcc $^ -o $@
```

Vamos executar:

```
1  $ ./exemplo2
2  -- 14
3  - 13
4  --- 12
5  -- 10
6   8
7  -- 7
8  ---- 5
9  --- 4
10 - 3
11 -- 1
12 > 10
13 REMOVIDO!!
14 ...
```



Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Rafael C. S. Schouery, da Universidade Estadual de Campinas.
2. Feofiloff, Paulo. Algoritmos em linguagem C. Elsevier Brasil, 2009.