Arbres bicolores

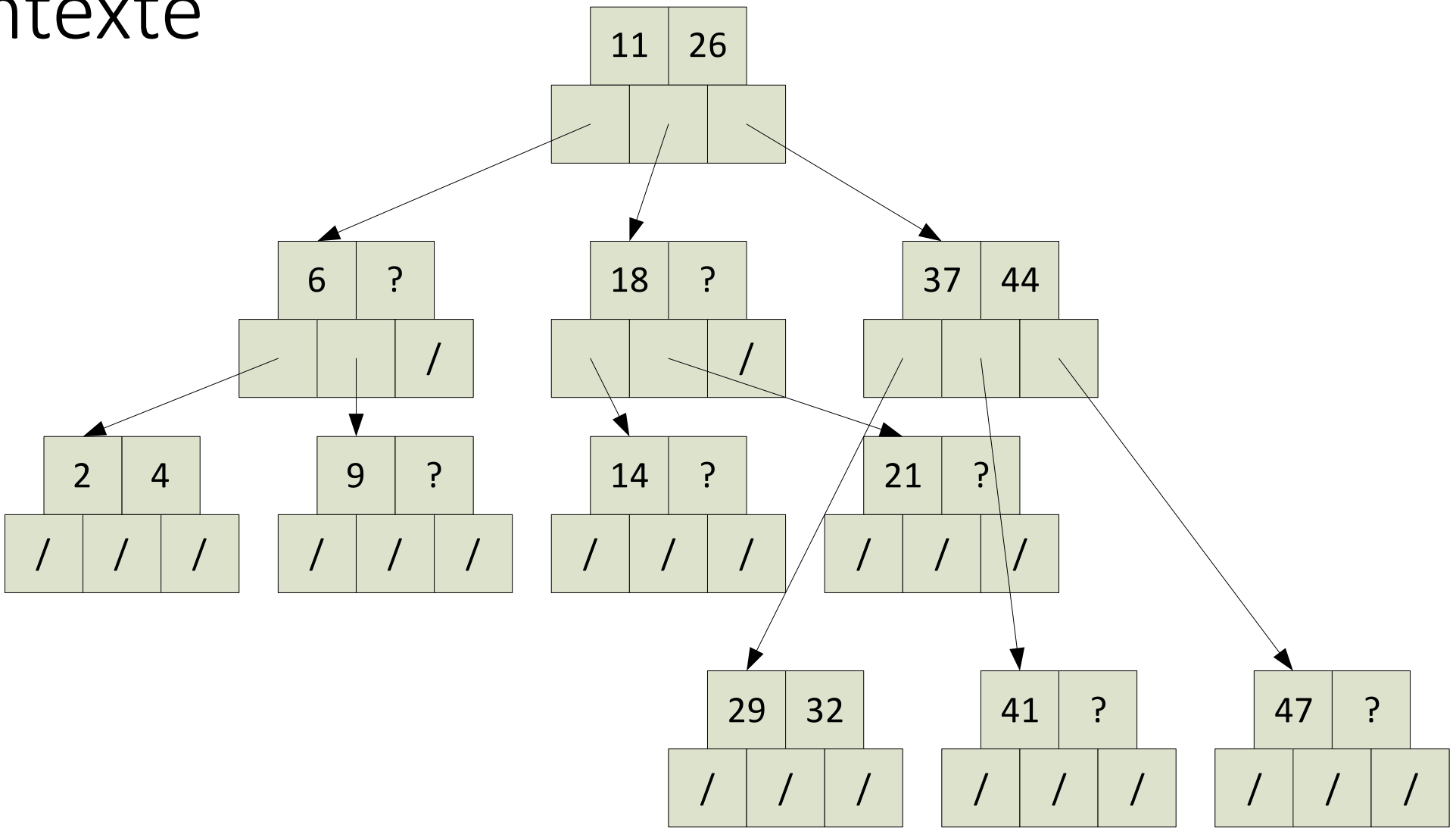
Mieux connus sous le nom d’arbres rouge-noir

Contexte

Contexte

- Supposons un B-Tree d’ordre 3 ou d’ordre 4

Contexte

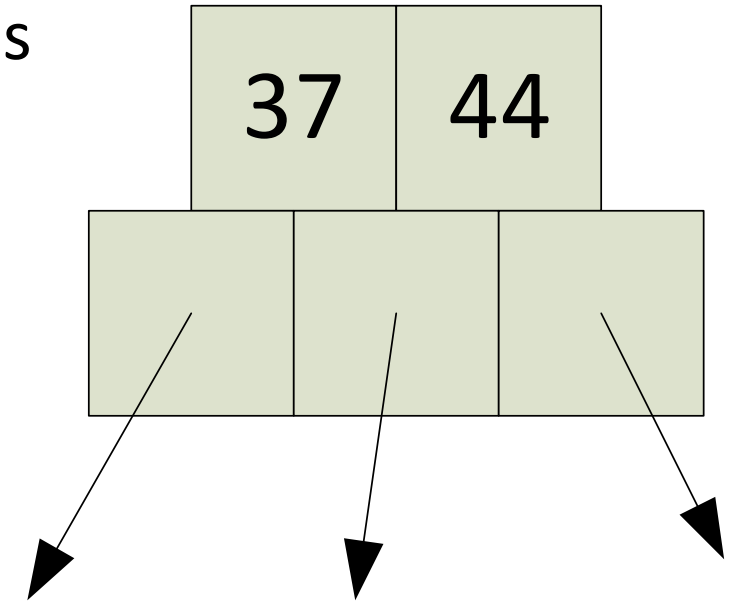


Contexte

- Supposons un B-Tree d’ordre 3 ou d’ordre 4
- Examinons de plus près l’un des noeuds

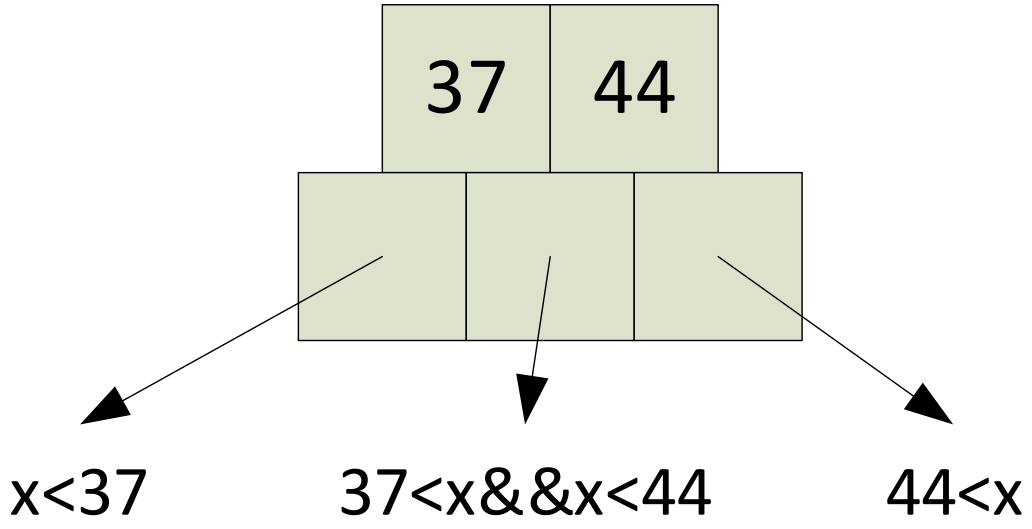
Contexte

- Supposons un B-Tree d’ordre 3 ou d’ordre 4
- Examinons de plus près l’un des noeuds



Contexte

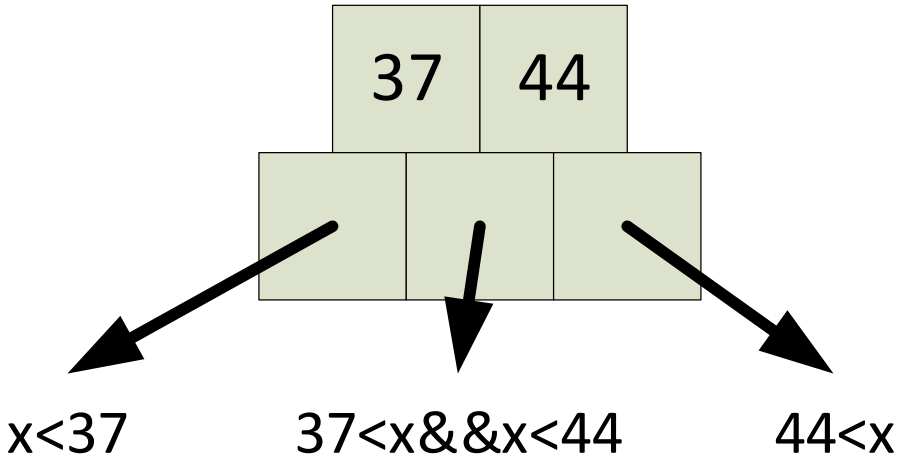
- Supposons un B-Tree d’ordre 3 ou d’ordre 4
- Examinons de plus près l’un des noeuds



Contexte

- Supposons un B-Tree d'ordre 3 ou d'ordre 4
- Examinons de plus près l'un des noeuds

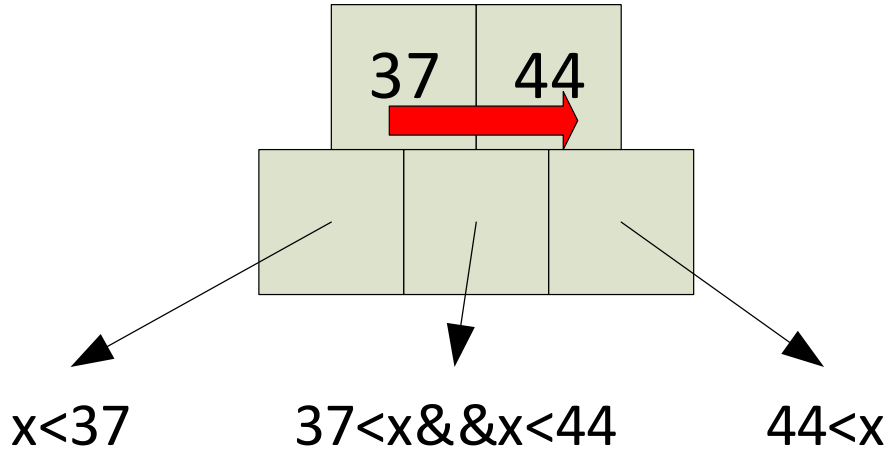
Transitions vers un autre niveau (vers les enfants), arc « noir »



Contexte

- Supposons un B-Tree d’ordre 3 ou d’ordre 4
- Examinons de plus près l’un des noeuds

Transitions dans le même niveau (interne au noeud), arc « rouge »



Contexte

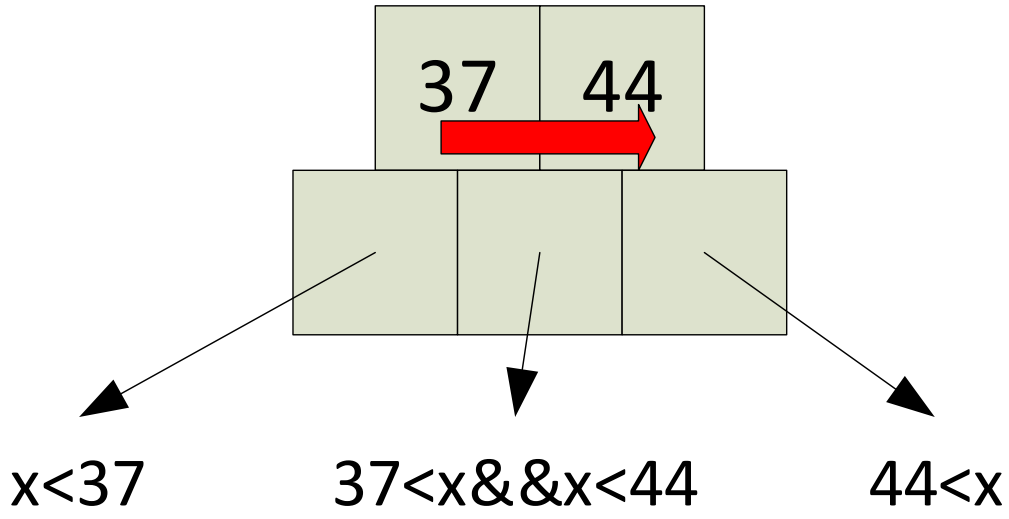
- Supposons un B-Tree d’ordre 3 ou d’ordre 4
- Examinons de plus près l’un des nœuds
- On peut scinder ce nœud en deux, à la manière de la scission faite dans un B-Tree quand un nœud est saturé
  - Pour maintenir la cohérence de l’arbre, il faut toutefois mémoriser la couleur des arcs
  - Pour un nœud d’un B-Tree d’ordre 3, deux scissions sont possibles et équivalentes (une implémentation choisira son camp)

Contexte

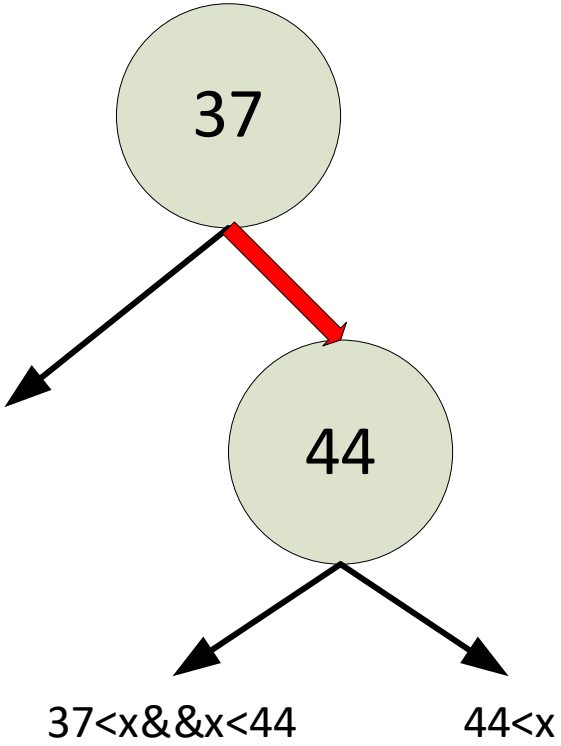
- Pour un nœud d’un B-Tree d’ordre 3, deux scissions sont possibles et équivalentes (une implémentation choisira son camp)

Contexte

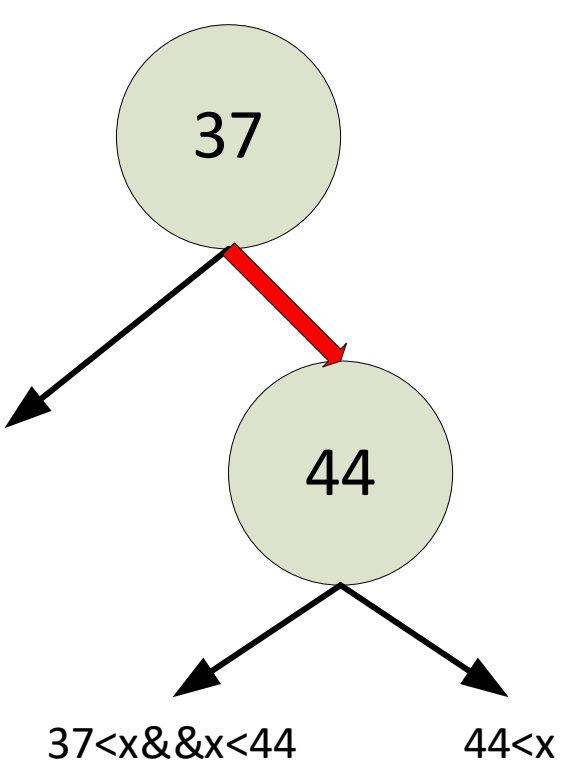
- Pour un nœud d'un B-Tree d'ordre 3, deux scissions sont possibles et équivalentes (une implémentation choisira son camp)

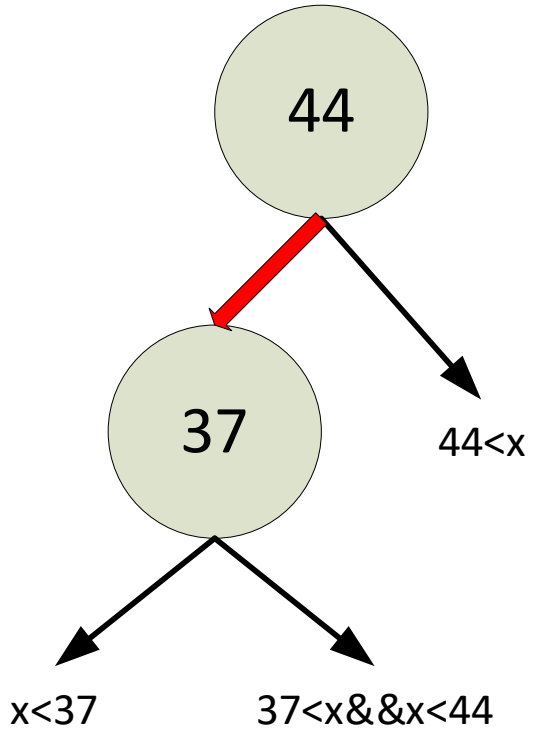


Contexte

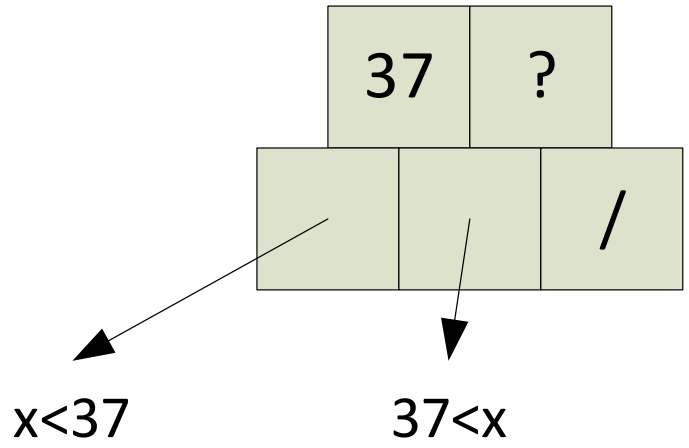


Contexte

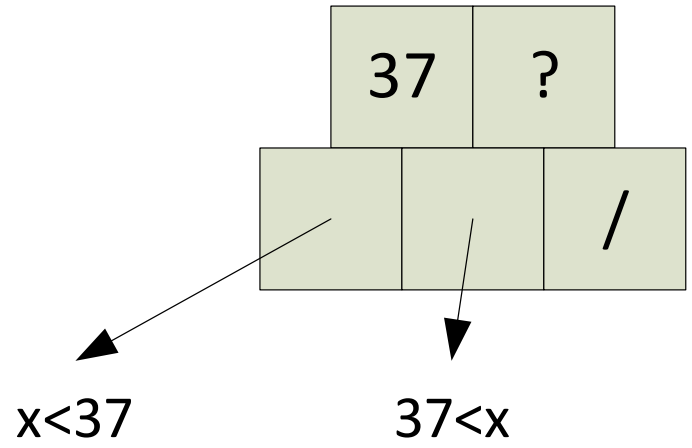




Contexte

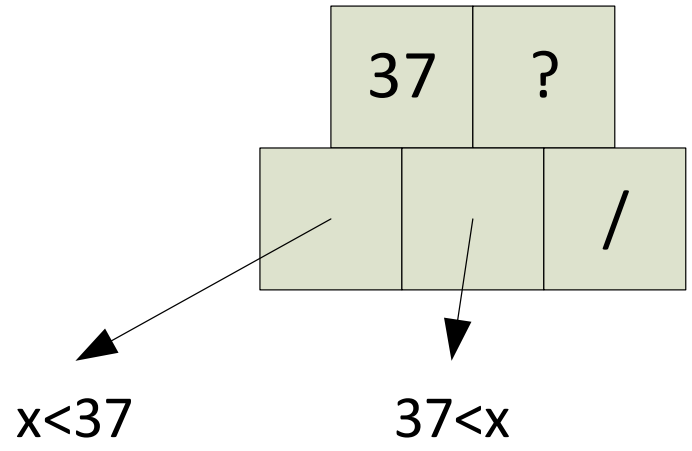


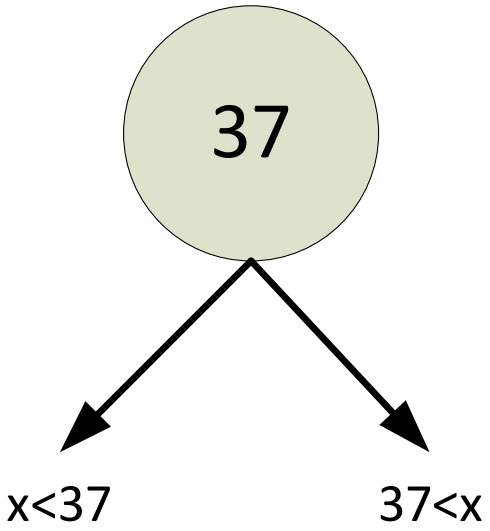
Contexte



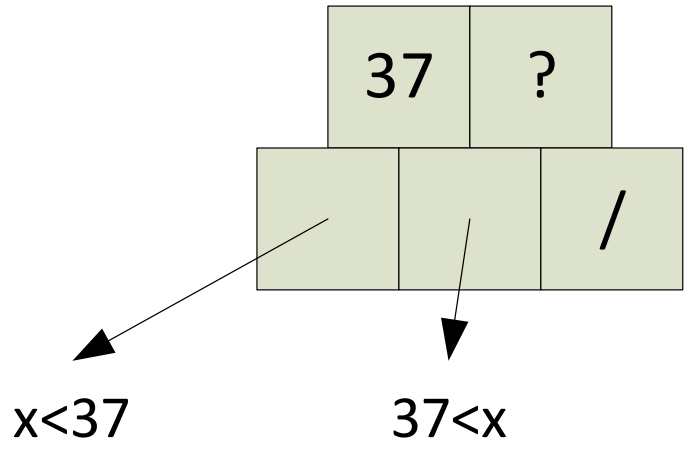
B-Tree d'ordre 3, mais avec un seul élément (équivalent à un nœud dans un BST)

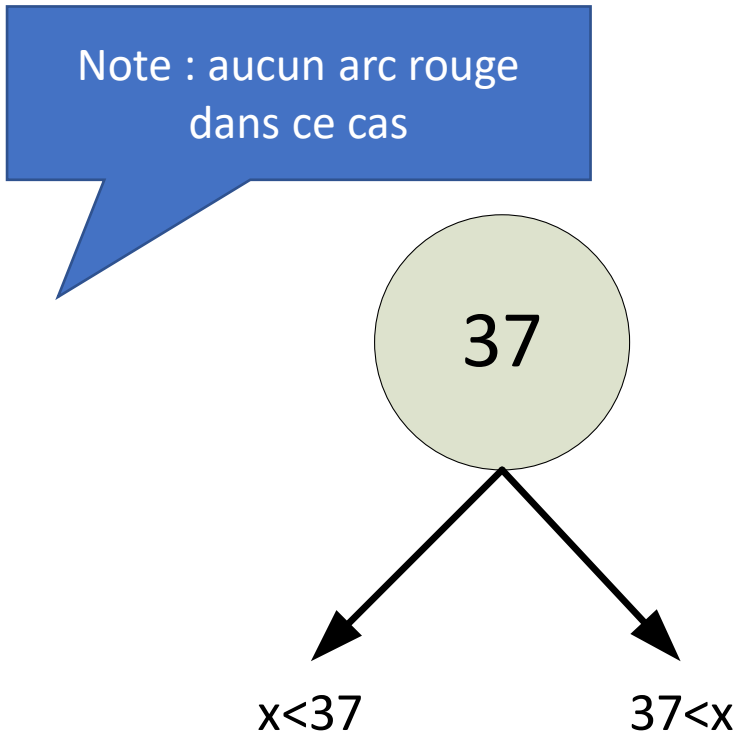
Contexte





Contexte

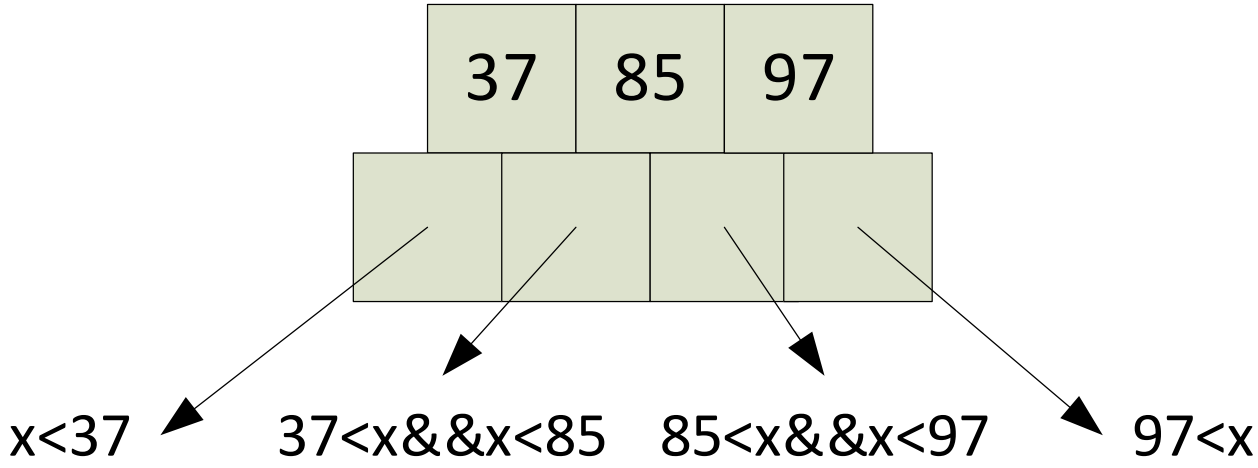




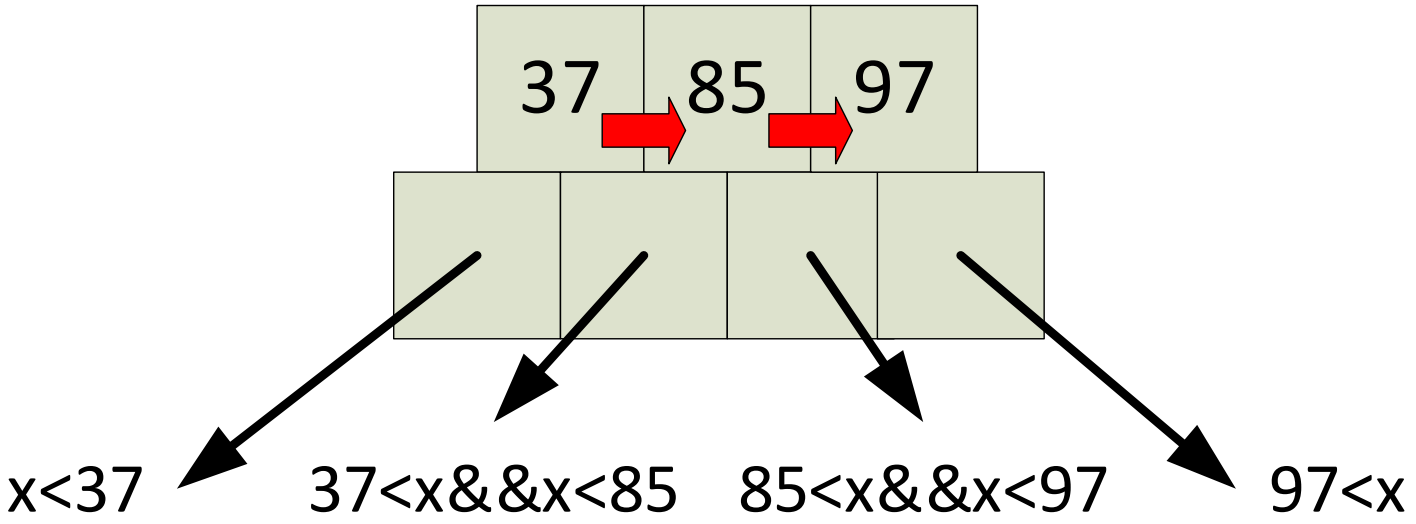
Contexte

- Pour un nœud d’un B-Tree d’ordre 3, deux scissions sont possibles et équivalentes (une implémentation choisira son camp)
- Pour un nœud d’un B-Tree d’ordre 4, une seul scission est possible

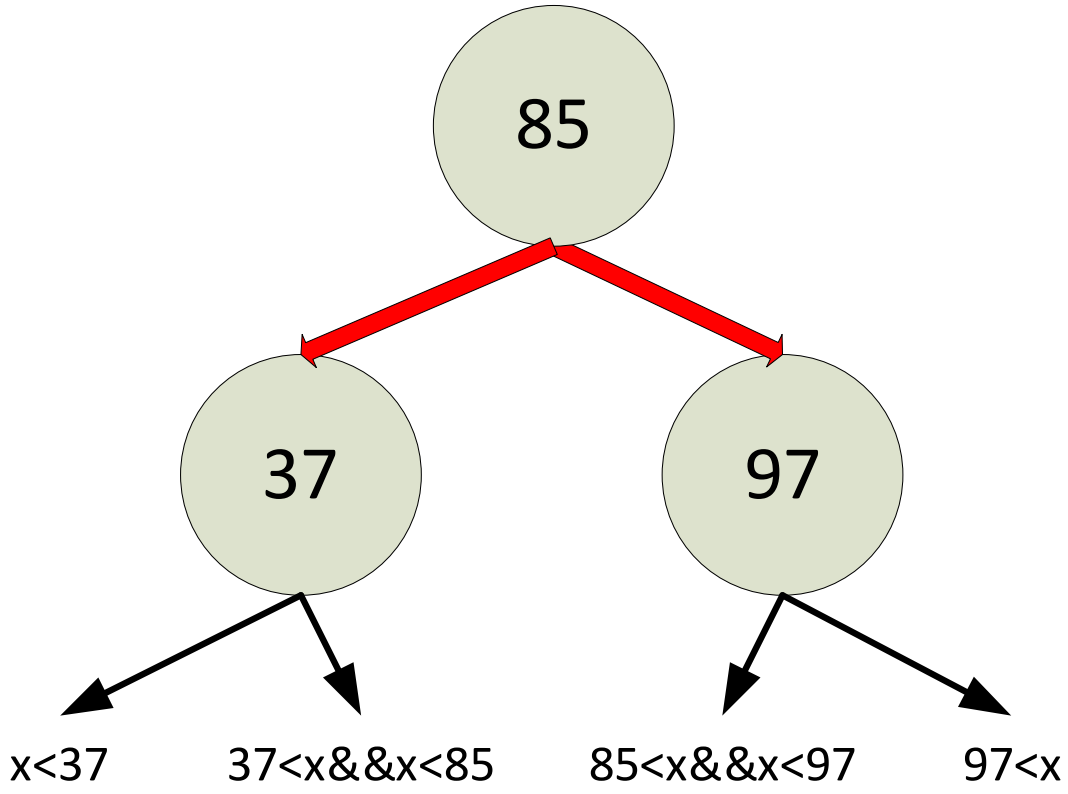
Contexte



Contexte



Contexte

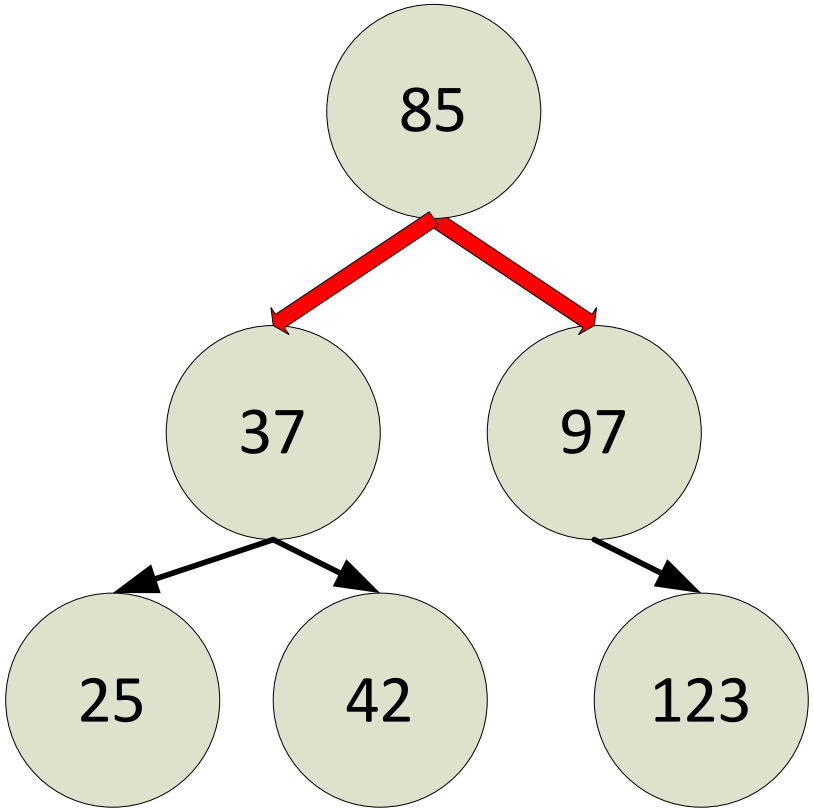


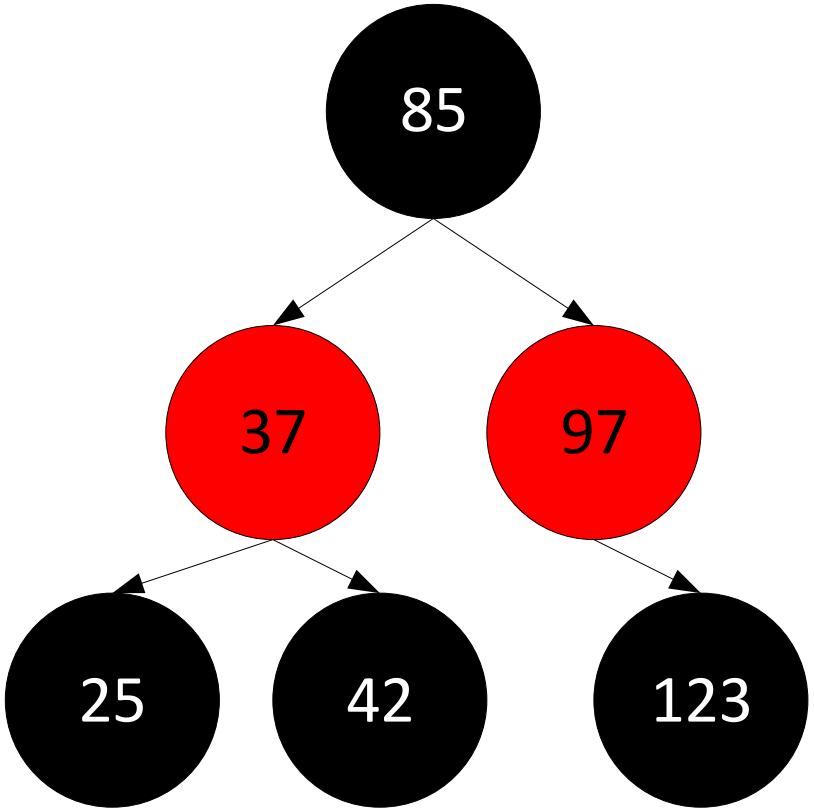
Définition

Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc est rouge ou noir (souvent, on dira que chaque nœud est rouge ou noir, mais on parle alors du nœud vers lequel mène l'arc)

Définition





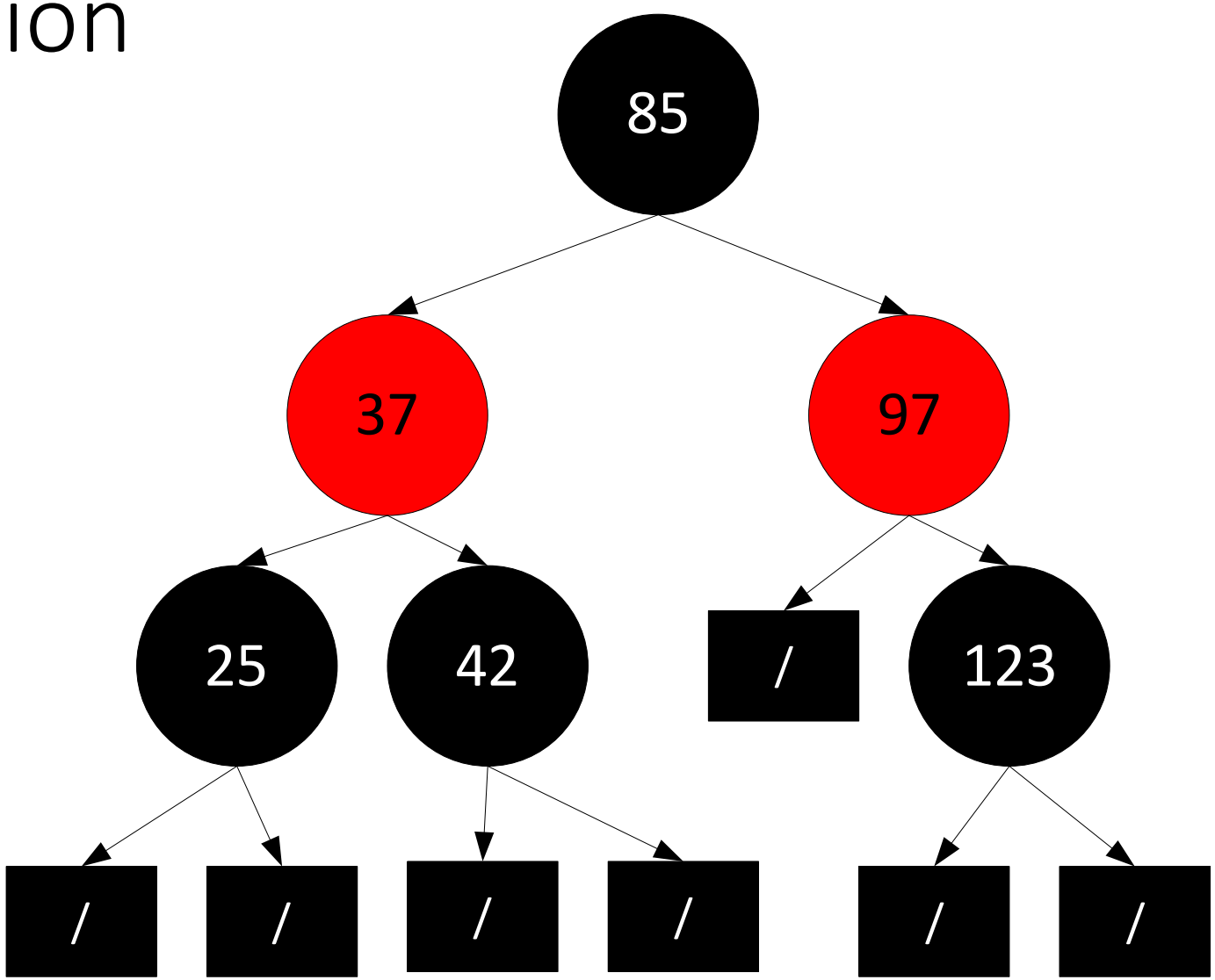
Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc / chaque noeud est rouge ou noir
  - La racine est un nœud noir

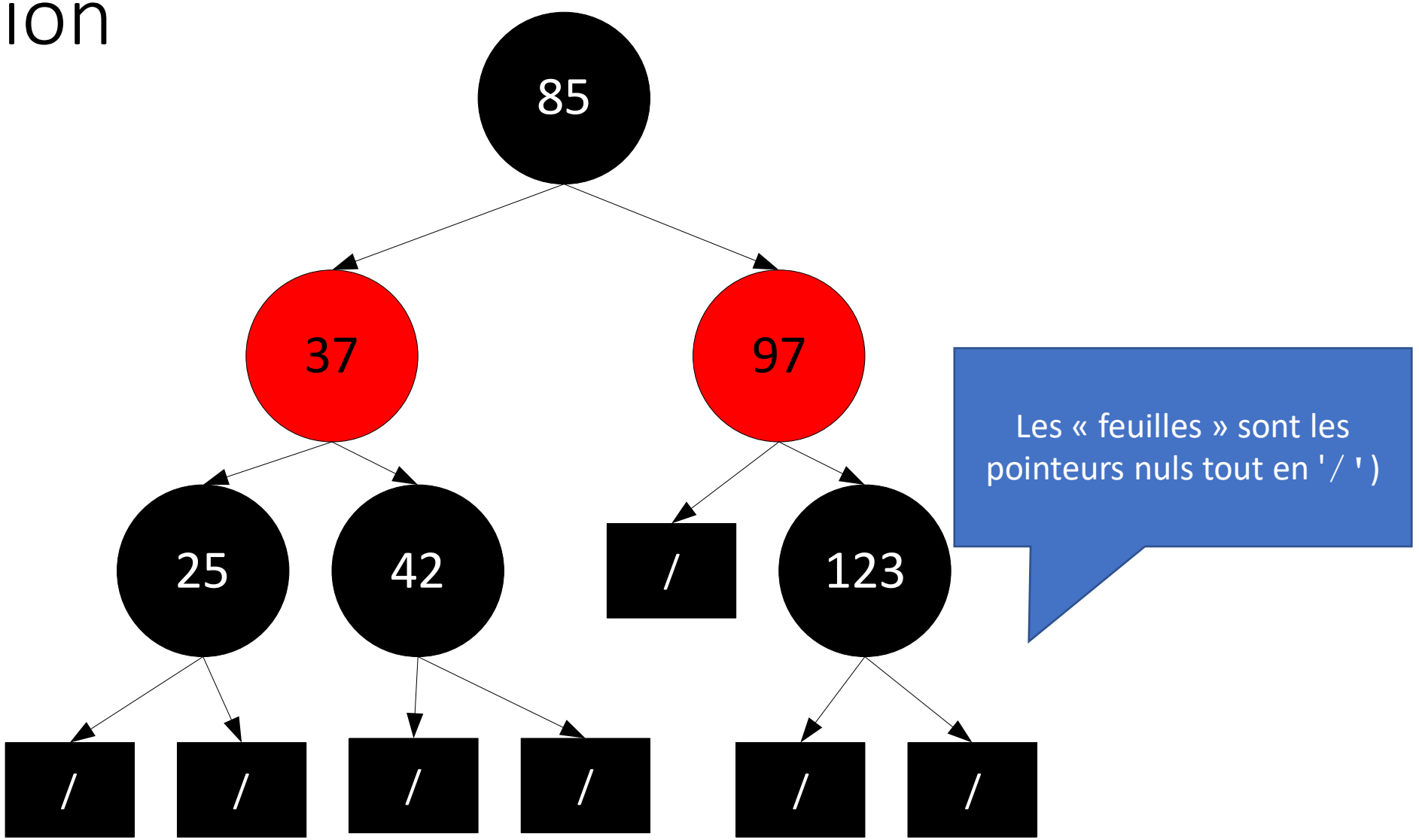
Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc / chaque noeud est rouge ou noir
  - La racine est un nœud noir
  - Les « feuilles » sont des nœuds noirs
    - Dans un arbre bicolore, les « feuilles » sont les pointeurs nuls sous les nœuds les plus bas, pas ces nœuds eux-mêmes

Définition



Définition


85
37
97
25
42
/
123
/
/
/
/
/
/
Les « feuilles » sont les pointeurs nuls tout en '/')

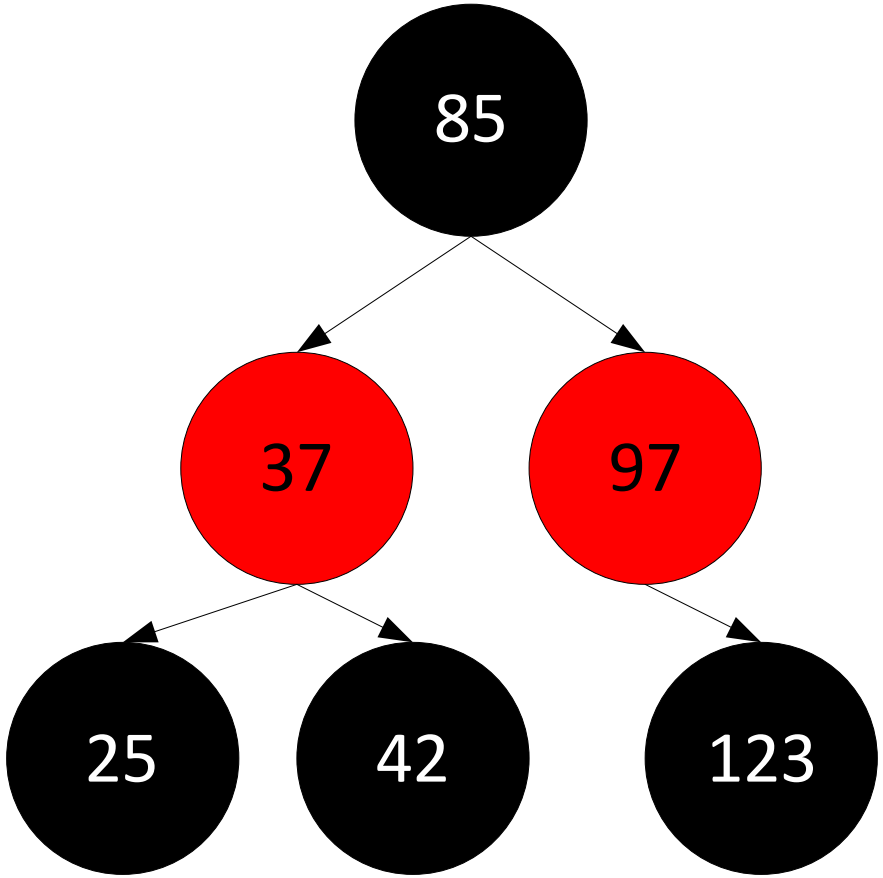
Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc / chaque noeud est rouge ou noir
  - La racine est un nœud noir
  - Les « feuilles » sont des nœuds noirs
  - Si un nœud est rouge, alors ses enfants sont noirs
    - Il n’y a donc jamais deux nœuds rouges consécutifs sur un même chemin

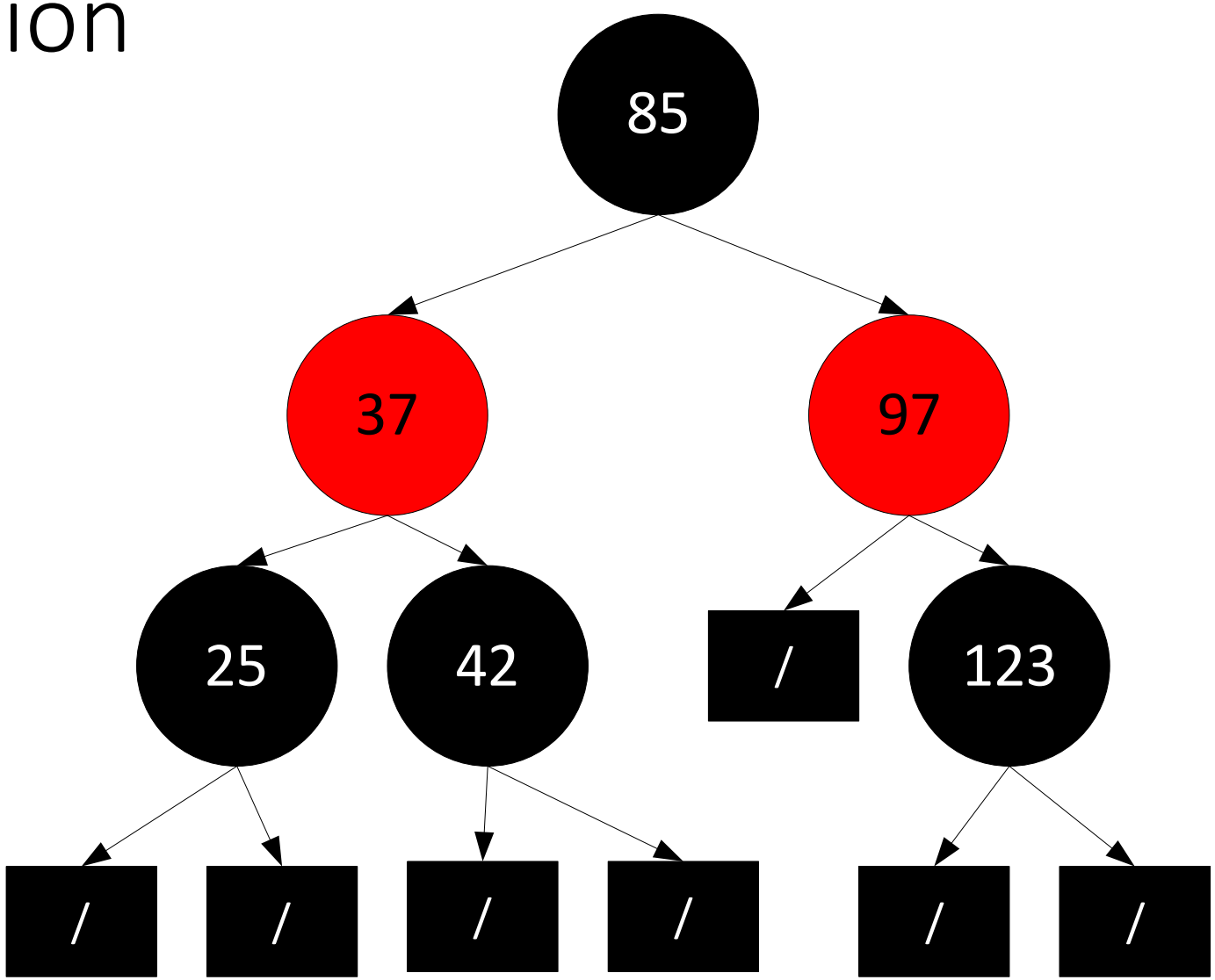
Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc / chaque noeud est rouge ou noir
  - La racine est un nœud noir
  - Les « feuilles » sont des nœuds noirs
  - Si un nœud est rouge, alors ses enfants sont noirs
  - Sur un chemin de la racine à une « feuille », il y a toujours le même nombre de nœuds noirs

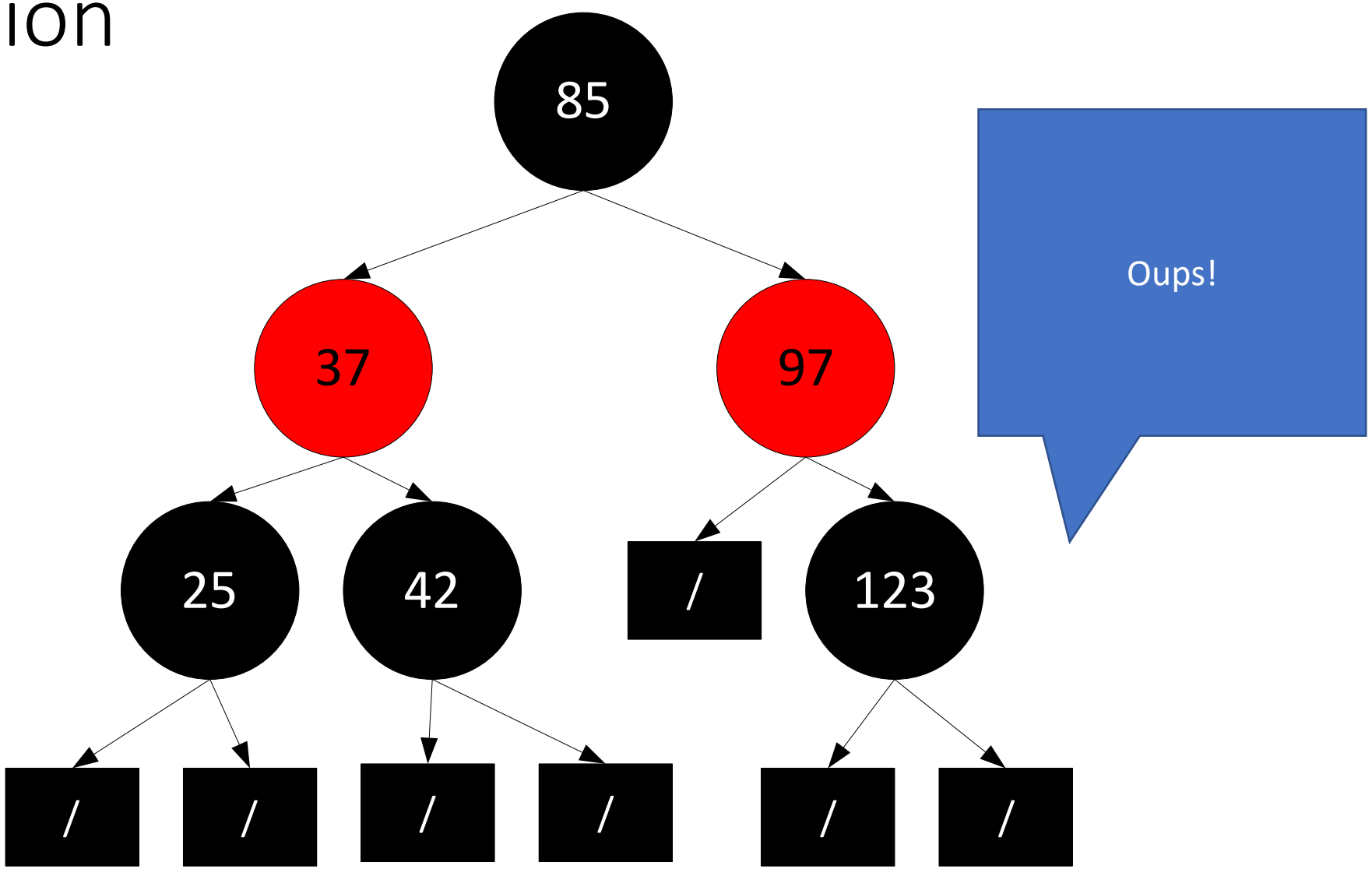
Définition



Définition



Définition



Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc / chaque noeud est rouge ou noir
  - La racine est un nœud noir
  - Les « feuilles » sont des nœuds noirs
  - Si un nœud est rouge, alors ses enfants sont noirs
  - Sur un chemin de la racine à une « feuille », il y a toujours le même nombre de nœuds noirs

Ce sont les invariants d'un arbre rouge-noir

Définition

- Un arbre bicolore, souvent appelé arbre rouge-noir, est un arbre binaire de recherche avec cinq propriétés supplémentaires
  - Chaque arc / chaque noeud est rouge ou noir
  - La racine est un nœud noir
  - Les « feuilles » sont des nœuds noirs
  - Si un nœud est rouge, alors ses enfants sont noirs
  - Sur un chemin de la racine à une « feuille », il y a toujours [illegible] de nœuds noirs

Si ces invariants sont brisés, notre BST n'est pas un arbre rouge-noir

Ce sont les invariants d'un arbre rouge-noir

Définition

- Un arbre bicolore, souvent app
  binaire de recherche avec cinq
  - Chaque arc / chaque noeud est r
  - La racine est un nœud noir
  - Les « feuilles » sont des nœuds noirs
  - Si un nœud est rouge, alors ses enfants sont noirs
  - Sur un chemin de la racine à une « feuille », il y a toujours
    de nœuds noirs

Sans surprises, les insertions ou les suppressions peuvent briser les invariants d'un arbre rouge-noir; conséquemment, ces invariants doivent être rétablis avant de conclure l'opération qui les a brisés

Si ces invariants sont brisés, notre BST n'est pas un arbre rouge-noir

Ce sont les invariants d'un arbre rouge-noir

Opérations de rééquilibrage

Opérations de rééquilibrage

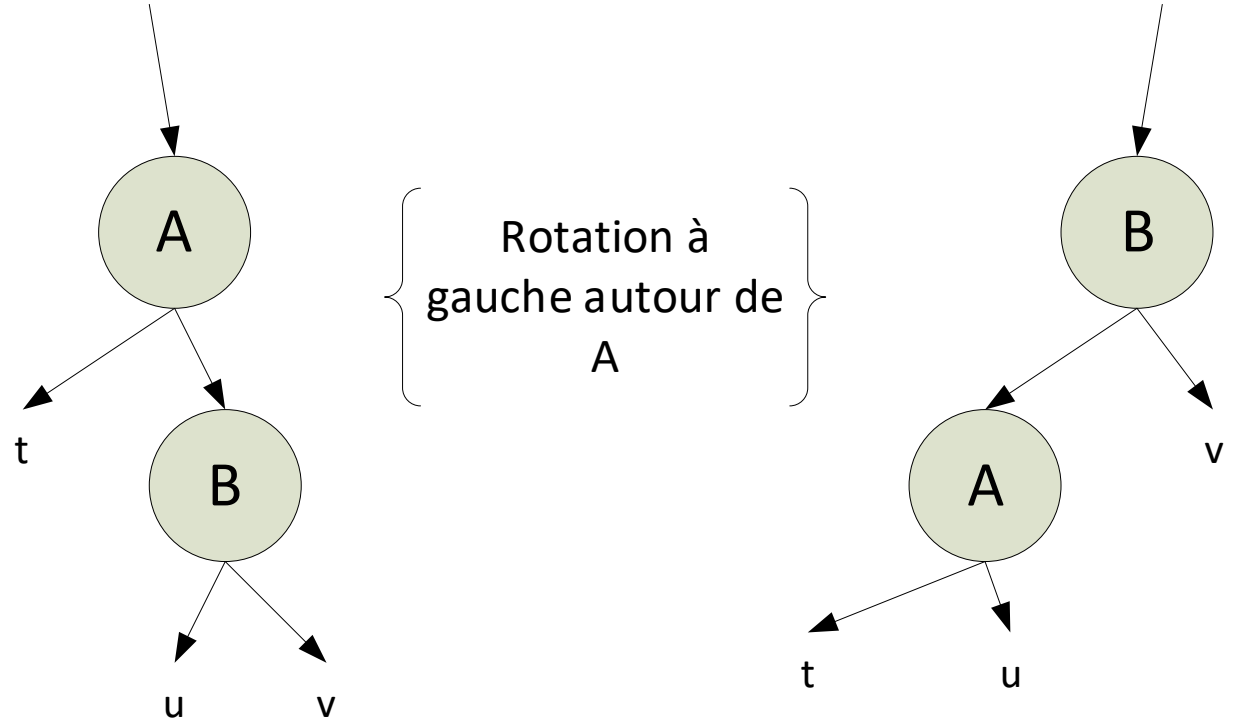
- Les opérations de rééquilibrage d’un arbre rouge noir sont :

Opérations de rééquilibrage

- Les opérations de rééquilibrage d’un arbre rouge noir sont :
  - La rotation à gauche

Opérations de rééquilibrage

- Les opérations de rééquilibrage d'un arbre rouge noir sont :
  - La rotation à gauche

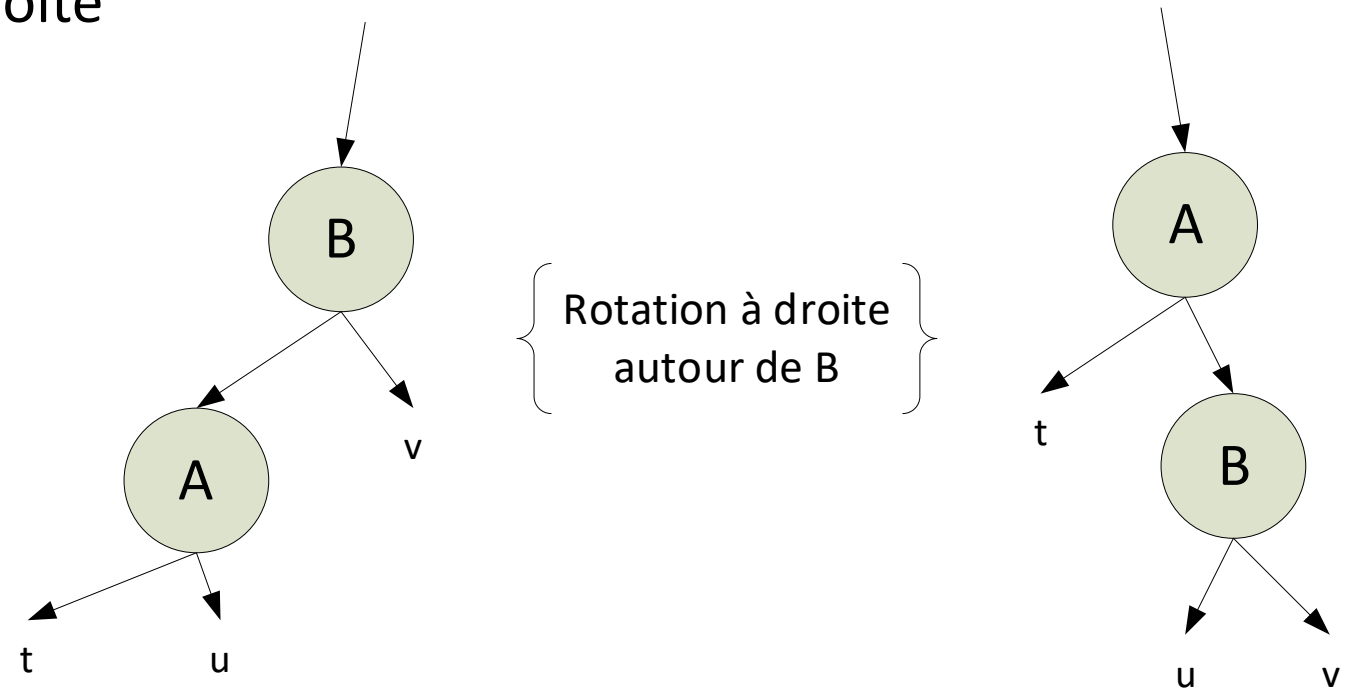


Opérations de rééquilibrage

- Les opérations de rééquilibrage d’un arbre rouge noir sont :
  - La rotation à gauche
  - La rotation à droite

Opérations de rééquilibrage

- Les opérations de rééquilibrage d'un arbre rouge noir sont :
  - La rotation à gauche
  - La rotation à droite



Opérations de rééquilibrage

- Les opérations de rééquilibrage d'un arbre rouge noir sont :
  - La rotation à gauche
  - La rotation à droite

Pas de choc jusqu'ici

Opérations de rééquilibrage

- Les opérations de rééquilibrage d’un arbre rouge noir sont :
  - La rotation à gauche
  - La rotation à droite
  - La recoloration
    - Passage d’un nœud de la couleur noire à la couleur rouge ou inversement

Autres opérations

Autres opérations

- Puisqu'un arbre rouge-noir est un BST, il offre la même gamme d'opérations que les autres arbres binaires de recherche
  - Pour les opérations immuables (méthodes const), ces opérations sont même implémentées de la même manière

Autres opérations

- Puisqu'un arbre rouge-noir est un BST, il offre la même gamme d'opérations que les autres arbres binaires de recherche
  - Pour les opérations immuables (méthodes const), ces opérations sont même implémentées de la même manière
- Deux différences notables, ces opérations mutatrices que sont l'insertion et la suppression

Insertion

Insertion

- L’idée générale est simple
  - Insérer normalement (trouver la feuille – parent de la « feuille » étant donné la définition de « feuille » pour un arbre rouge-noir) le nouveau nœud

Insertion

- L’idée générale est simple
  - Insérer normalement (trouver la feuille – parent de la « feuille » étant donné la définition de « feuille » pour un arbre rouge-noir) le nouveau nœud
  - Le colorier rouge

Insertion

- L’idée générale est simple
  - Insérer normalement (trouver la feuille – parent de la « feuille » étant donné la définition de « feuille » pour un arbre rouge-noir) le nouveau nœud
  - Le colorier rouge
  - Vérifier que les invariants sont toujours respectés

Insertion

- L’idée générale est simple
  - Insérer normalement (trouver la feuille – parent de la « feuille » étant donné la définition de « feuille » pour un arbre rouge-noir) le nouveau nœud
  - Le colorier rouge
  - Vérifier que les invariants sont toujours respectés
  - S’ils ne le sont plus, corriger la situation

Insertion

- Supposons que l’on insère la valeur *val* dans l’arbre
- Les cas possibles sont au nombre de six
  - … mais c’est en fait trois « grands » cas et deux « sous-cas »

Insertion

- Cas A : l’arbre est vide au moment de l’insertion
  - Dans ce cas, le nœud contenant val devient la racine, et on le colorie en noir

Insertion

- Cas B : le parent du nouveau nœud est noir
  - Dans ce cas, il n’y a rien à faire, les invariants devraient être respectés

Insertion

- Cas C : le parent du nouveau nœud est rouge
  - Un invariant est brisé (deux nœuds rouges se suivent sur un même chemin)
  - On sait que le « grand-parent » est noir
    - Par définition, puisque le parent est rouge
  - On examine l'« oncle » (le nœud frère ou sœur du parent)

Insertion

- Cas C : le parent du nouveau nœud est rouge (suite)
  - Si l'oncle est rouge, alors
    - On colorie le parent en noir
    - On colorie l'« oncle » en noir
    - On colorie le grand-parent en rouge (sauf s'il s'agit de la racine!)

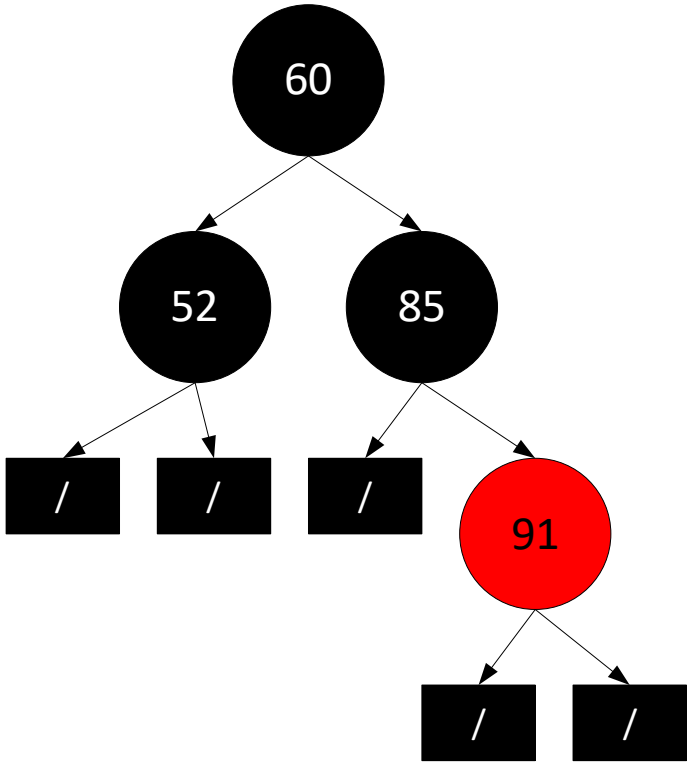
Insertion

- Cas C : le parent du nouveau nœud est rouge (suite)
  - Si l'oncle est noir (ou nul), alors
    - On aura une ou deux rotations à faire, selon le cas
    - Cas où le parent est enfant de *droite* du grand-parent et où val est enfant de *droite* du parent
    - Cas où le parent est enfant de *droite* du grand-parent et val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et où val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et val est enfant de *droite* du parent

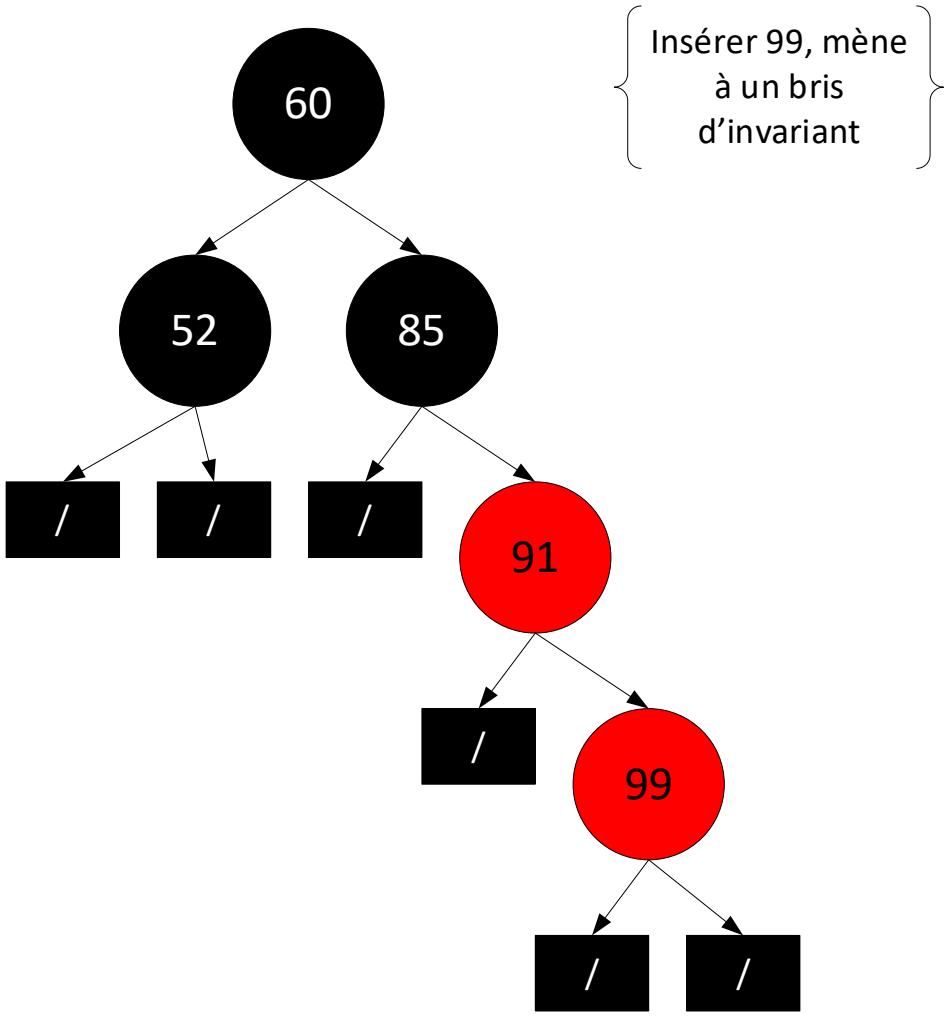
Insertion

- Cas C : le parent du nouveau nœud est rouge (suite)
  - Si l'oncle est noir (ou nul), alors
    - On aura une ou deux rotations à faire, selon le cas
    - **Cas où le parent est enfant de *droite* du grand-parent et où val est enfant de *droite* du parent**
    - Cas où le parent est enfant de *droite* du grand-parent et val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et où val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et val est enfant de *droite* du parent

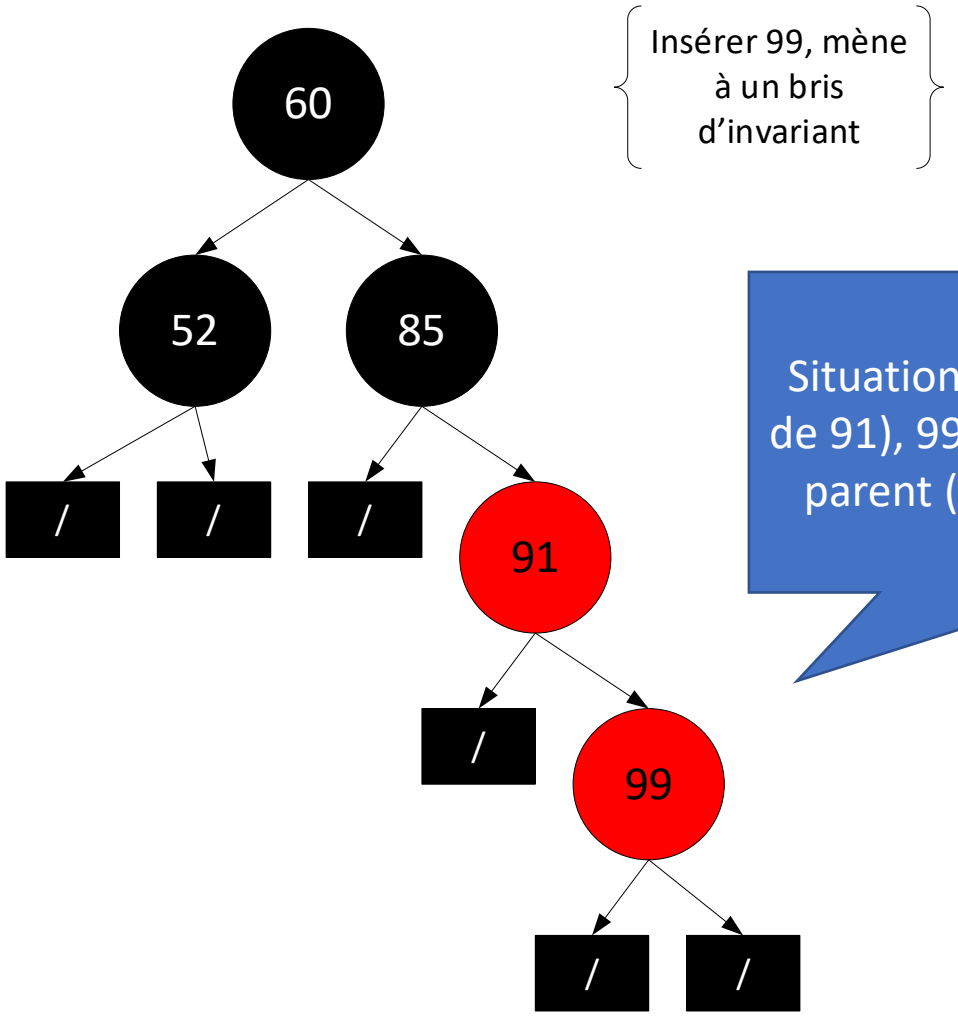
Insertion



Insertion



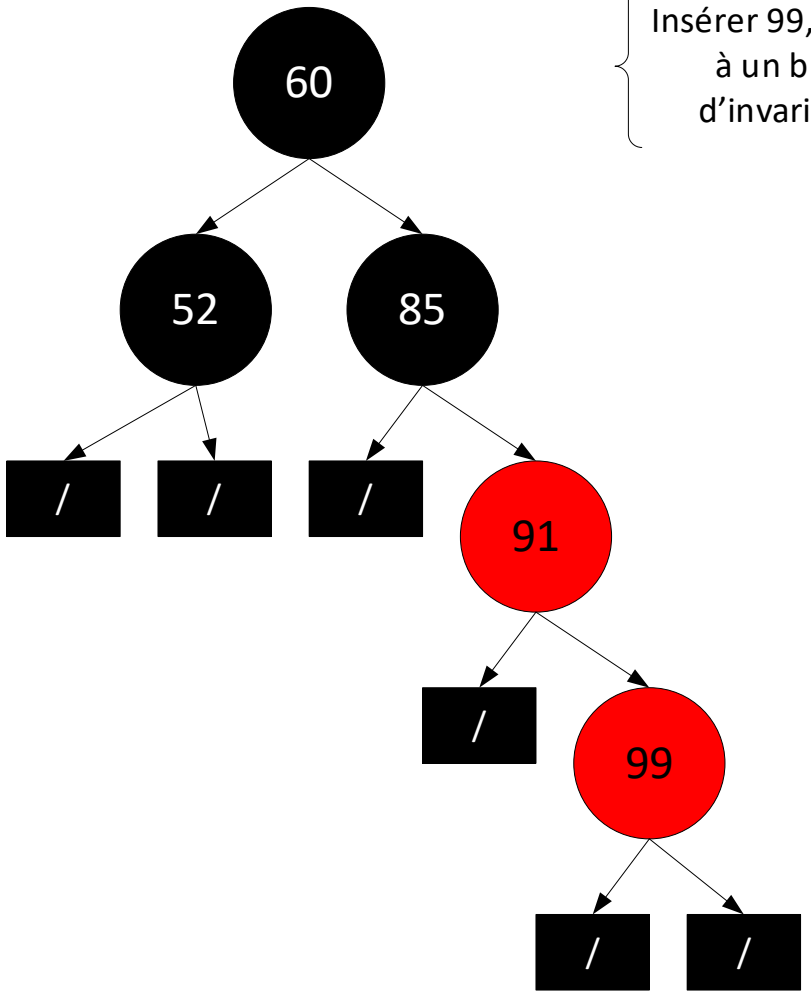
Insertion



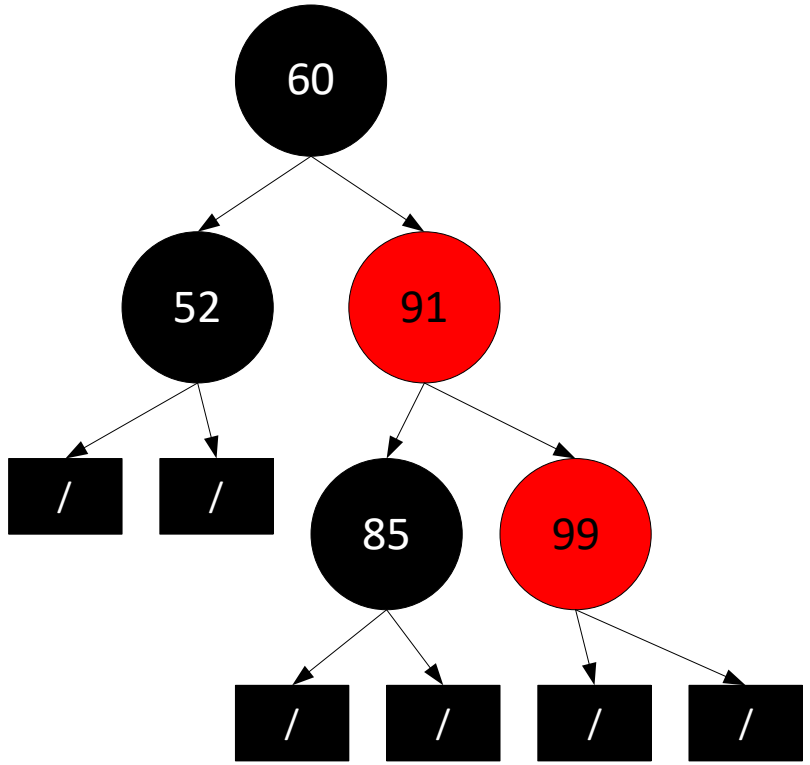
Insérer 99, mène à un bris d'invariant

Situation : oncle noir (enfant gauche de 91), 99 est un enfant de droite, son parent (91) est un enfant de droite

Insertion

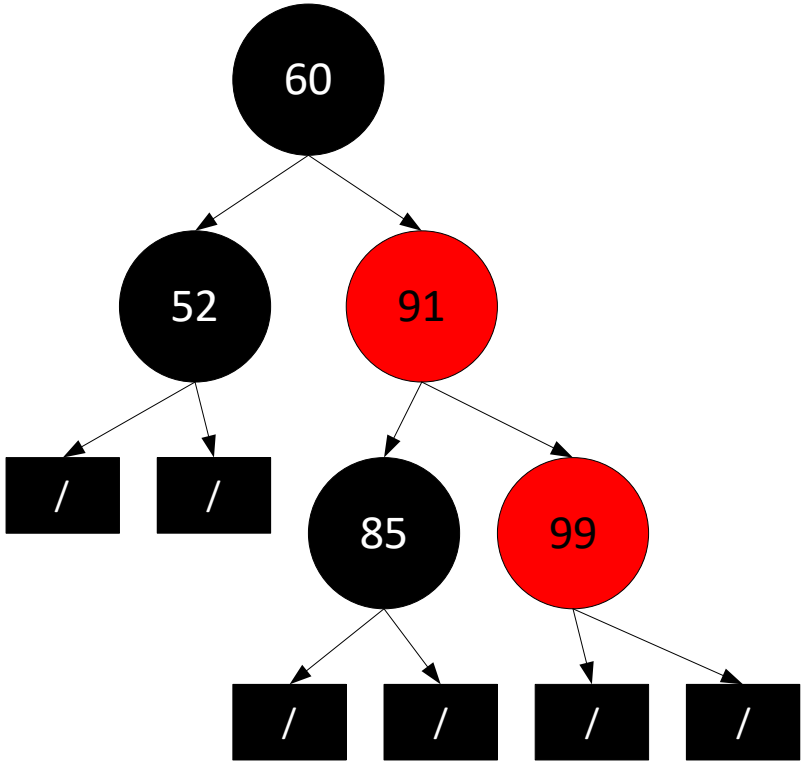


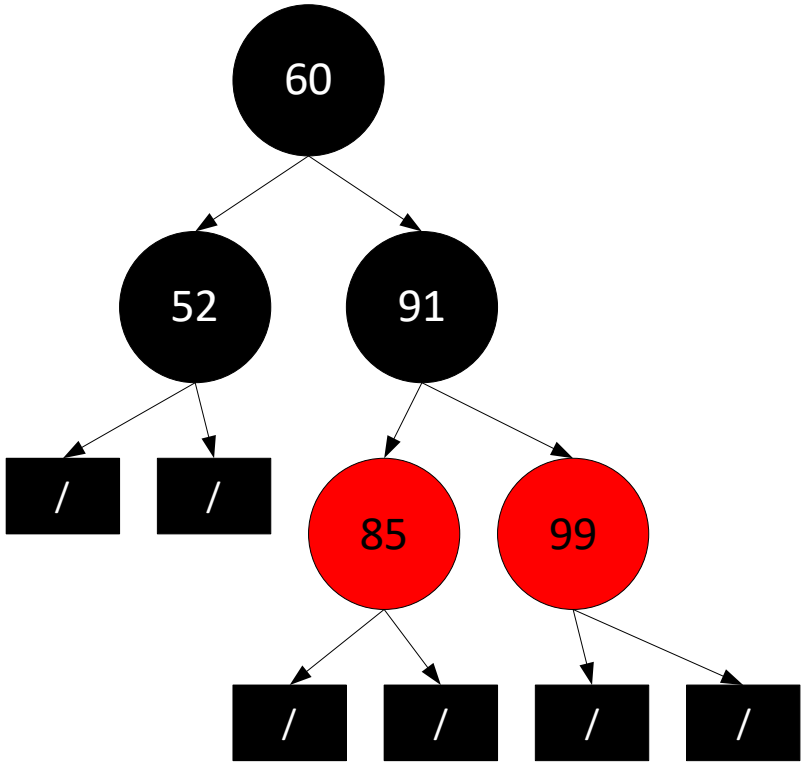
Insérer 99, mène à un bris d'invariant



Rotation de gauche autour du grand-parent (85)

Insertion



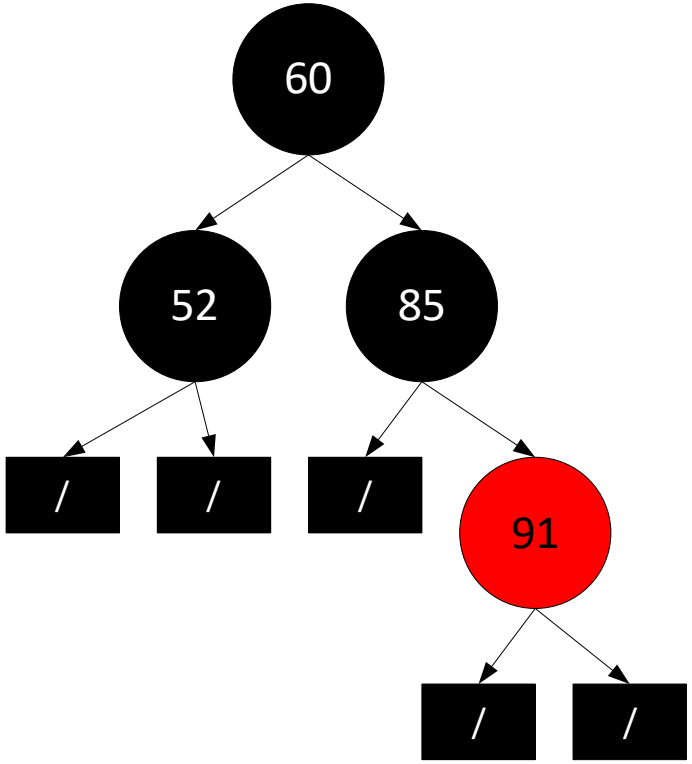


Puisque nous ne sommes toujours pas un arbre rouge-noir, on recolorie le parent et le frère

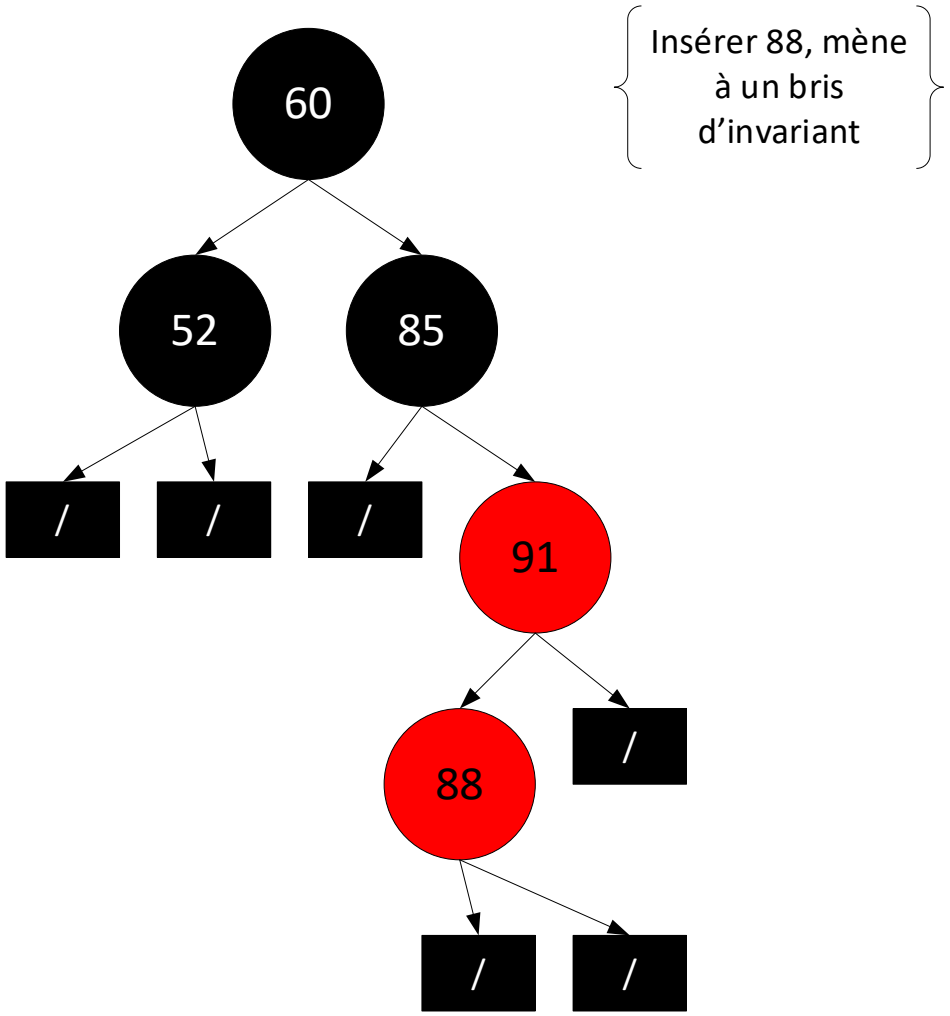
Insertion

- Cas C : le parent du nouveau nœud est rouge (suite)
  - Si l'oncle est noir (ou nul), alors
    - On aura une ou deux rotations à faire, selon le cas
    - Cas où le parent est enfant de *droite* du grand-parent et où val est enfant de *droite* du parent
    - **Cas où le parent est enfant de *droite* du grand-parent et val est enfant de *gauche* du parent**
    - Cas où le parent est enfant de *gauche* du grand-parent et où val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et val est enfant de *droite* du parent

Insertion

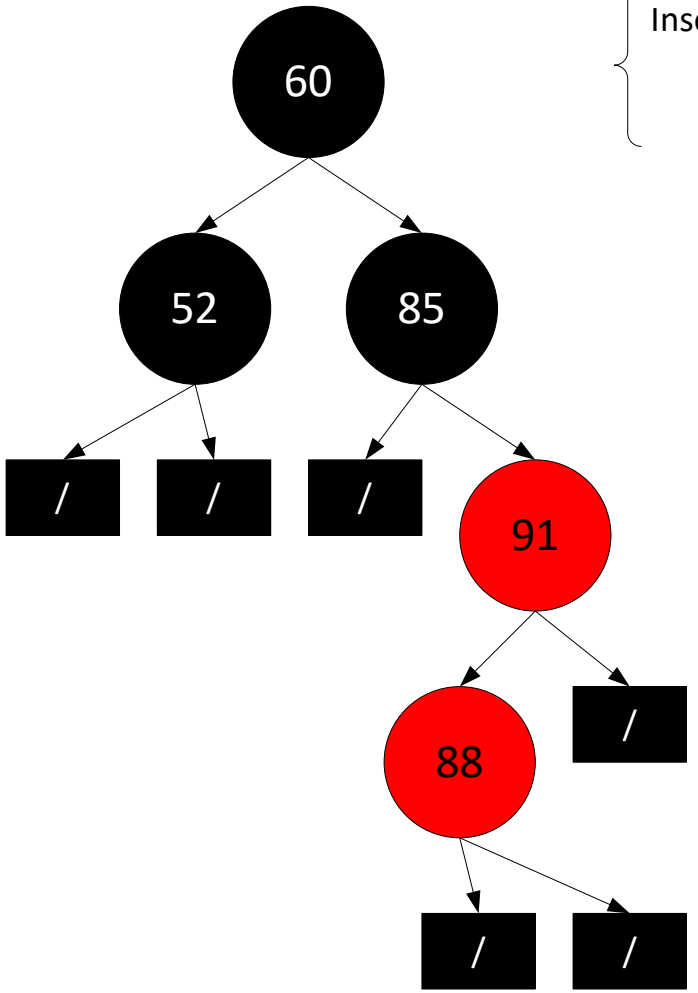


Insertion



Insérer 88, mène à un bris d'invariant

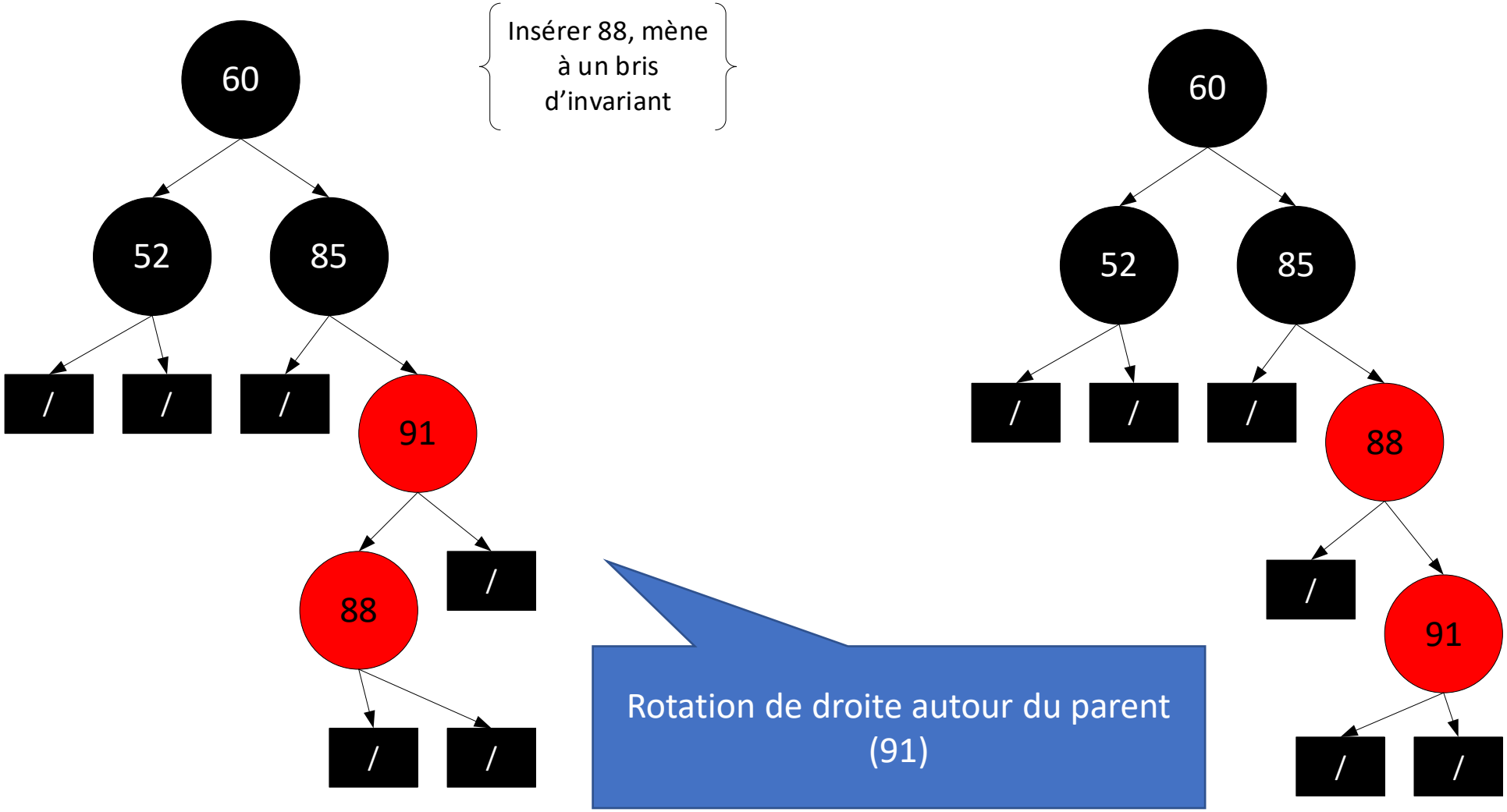
Insertion



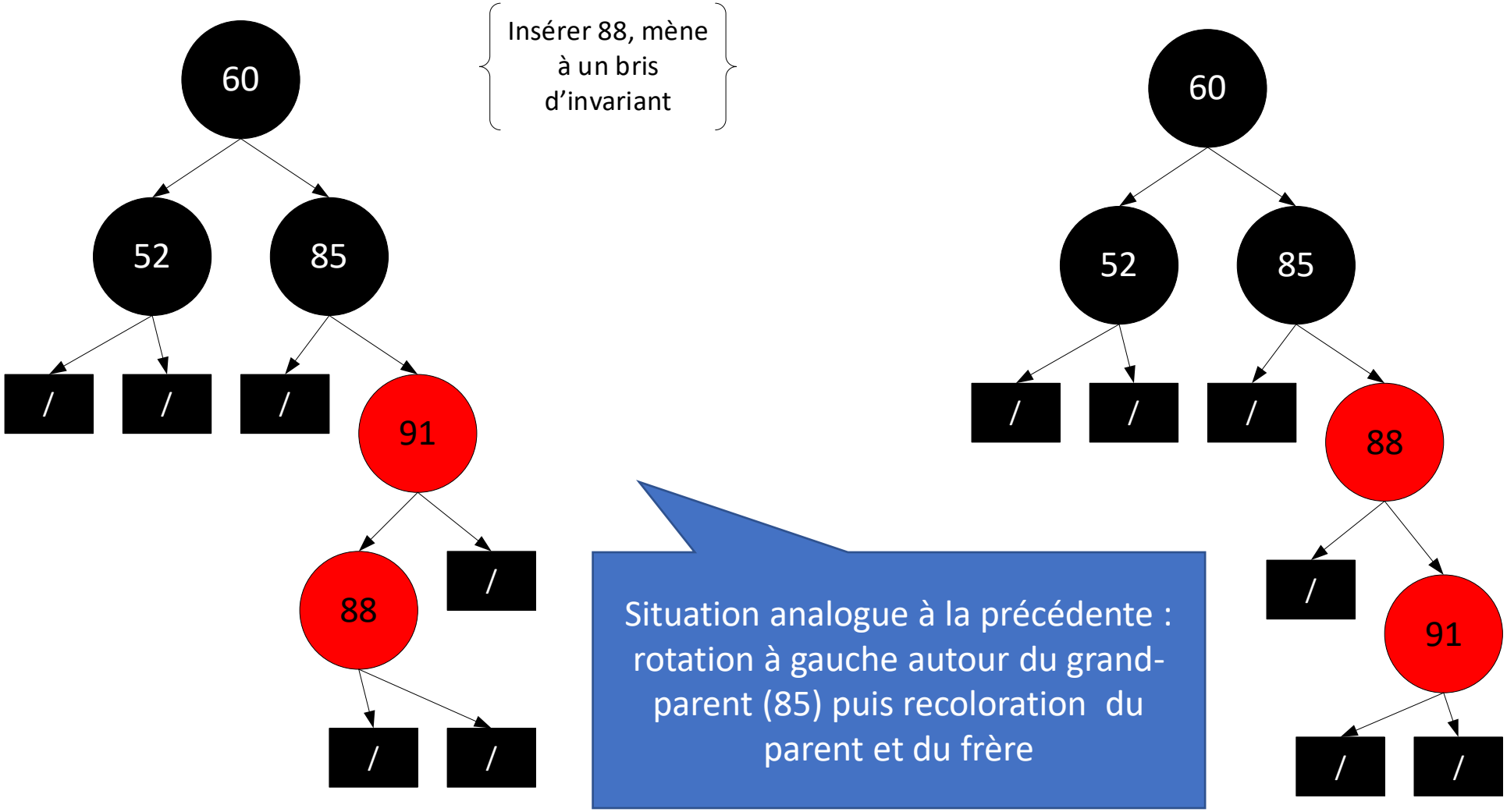
Insérer 88, mène à un bris d'invariant

Situation : oncle noir (enfant droit de 91), 88 est un enfant de gauche, son parent (91) est un enfant de droite

Insertion


Insérer 88, mène à un bris d'invariant
60
52
85
/
/
/
91
88
/
/
/
Rotation de droite autour du parent (91)
60
52
85
/
/
/
88
/
91
/
/

Insertion


Insérer 88, mène à un bris d'invariant
60
52
85
/
/
/
91
88
/
/
/
Situation analogue à la précédente : rotation à gauche autour du grand-parent (85) puis recoloration du parent et du frère
60
52
85
/
/
/
88
/
91
/
/

Insertion

- Cas C : le parent du nouveau nœud est rouge (suite)
  - Si l'oncle est noir (ou nul), alors
    - On aura une ou deux rotations à faire, selon le cas
    - Cas où le parent est enfant de *droite* du grand-parent et où val est enfant de *droite* du parent
    - Cas où le parent est enfant de *droite* du grand-parent et val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et où val est enfant de *gauche* du parent
    - Cas où le parent est enfant de *gauche* du grand-parent et val est enfant de *droite* du parent

Ces cas sont les mêmes que les deux précédents (c'est symétrique)

Suppression

Suppression

- L’idée générale va comme suit
  - Supprimer « normalement » de l’arbre, comme pour d’autres BST
  - Faire quelques ajustements

Suppression

- L’idée générale va comme suit
  - Supprimer « normalement » de l’arbre, comme pour d’autres BST
  - Faire quelques ajustements

Rappel : pour supprimer normalement, on trouve le nœud à supprimer; si c’est une feuille, on l’enlève, sinon on permute la valeur du nœud avec celle de la feuille appropriée (ensuite, selon le type d’arbre, on rebalance)

Suppression

- Cas A : le nœud supprimé est rouge
  - Rien de spécial à faire

Suppression

- Cas B : le nœud supprimé a un enfant rouge
  - On remplace la valeur du nœud supprimé par la valeur du nœud enfant
  - On colorie le nouvel enfant en rouge
  - Ça suffit

Suppression

- Cas C : le nœud supprimé est noir
  - Ceci brise un invariant de l'arbre rouge-noir
    - Il y aura plus de nœuds noirs sur certains chemins que d'autres
    - Cet invariant doit être rétabli
  - La correction commence en « supposant » que le nœud *x* qui remplace le nœud supprimé a « un noir supplémentaire »
    - Ceci place *x* dans une situation … obscure
    - x est-il doublement noir ou à la fois noir et rouge?
  - Ensuite, on corrige du nœud *x* en remontant vers la racine

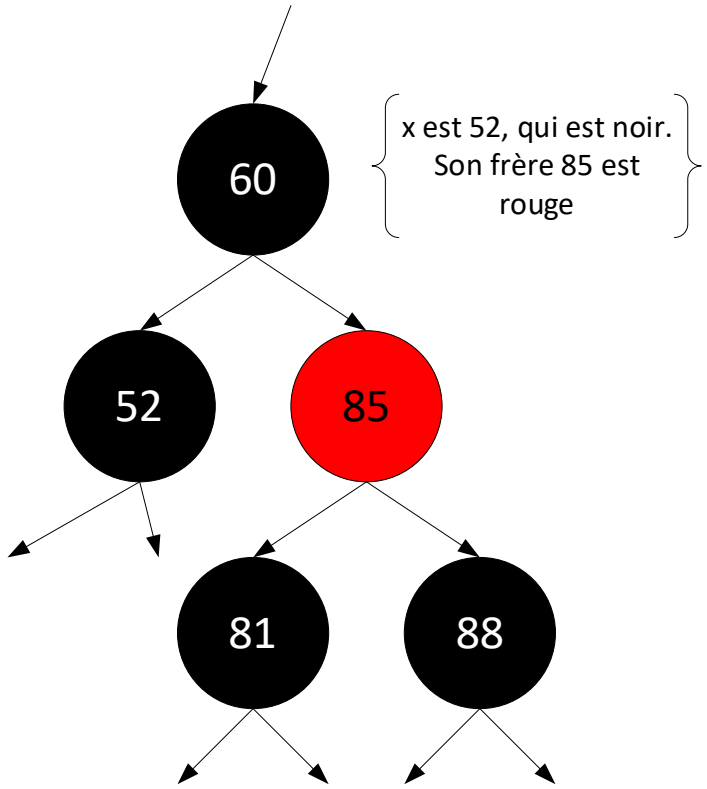
Suppression

- Cas C : le nœud supprimé est noir
  - Il y a des cas à explorer (huit, mais en fait quatre et leur équivalent symétrique… et plusieurs convergent vers d'autres cas)
    - C.1) Cas où le frère de *x* est rouge
    - C.2) Cas où le frère de *x* est noir et les enfants du frère sont tous deux noirs
    - C.3) Cas où le frère de *x* est noir, l'enfant de gauche du frère est rouge et l'enfant de droite du frère est noir
    - C.4) Cas où le frère de *x* est noir, l'enfant de gauche du frère est noir et l'enfant de droite du frère est rouge

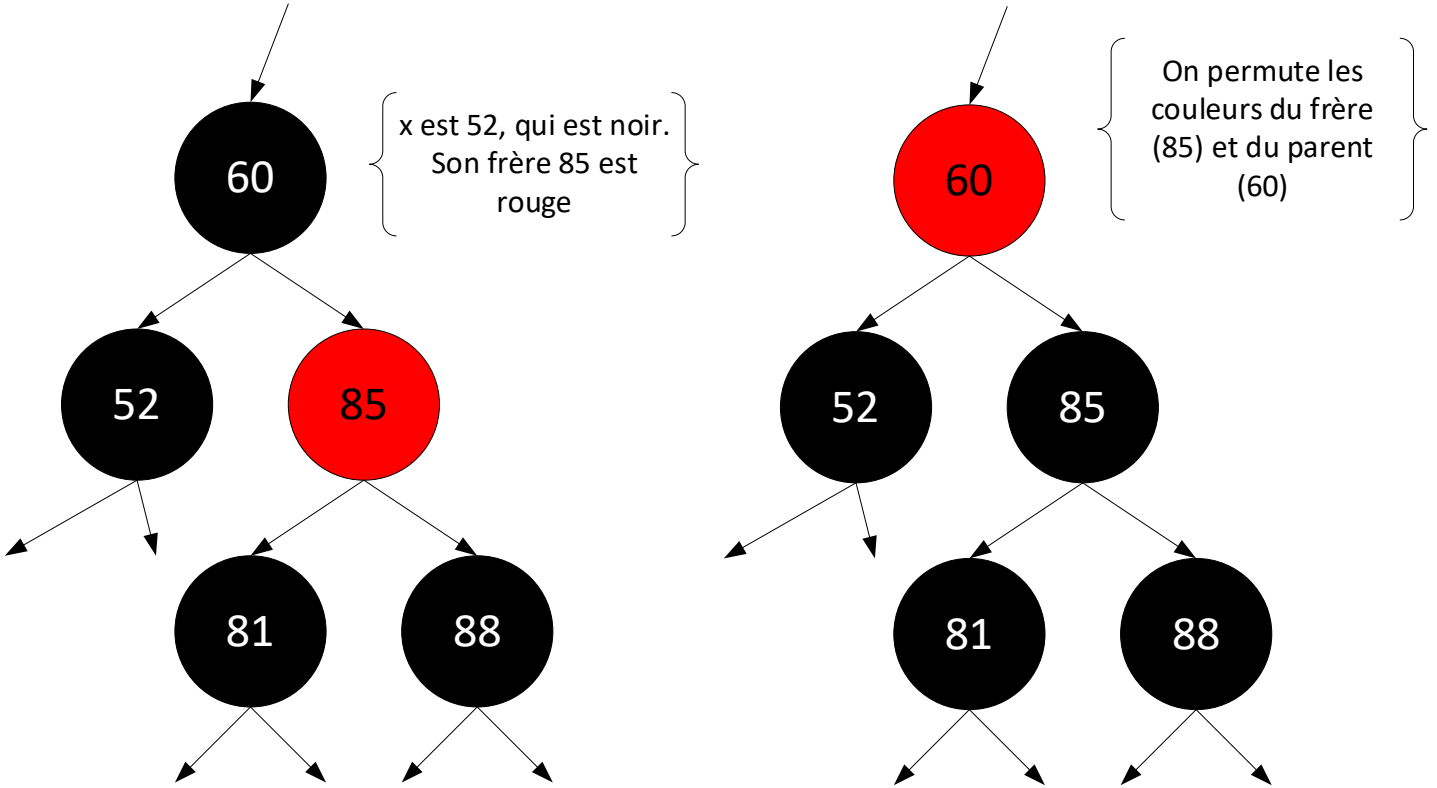
Suppression

- Cas C.1) le nœud supprimé est noir, le frère de *x* est rouge
  - On commence par deux transformations
    - Permuter les couleurs du frère et du parent
    - Rotation à gauche autour du parent
  - Ceci nous amène dans l'un des trois autres cas que sont C.2), C.3) et C.4)

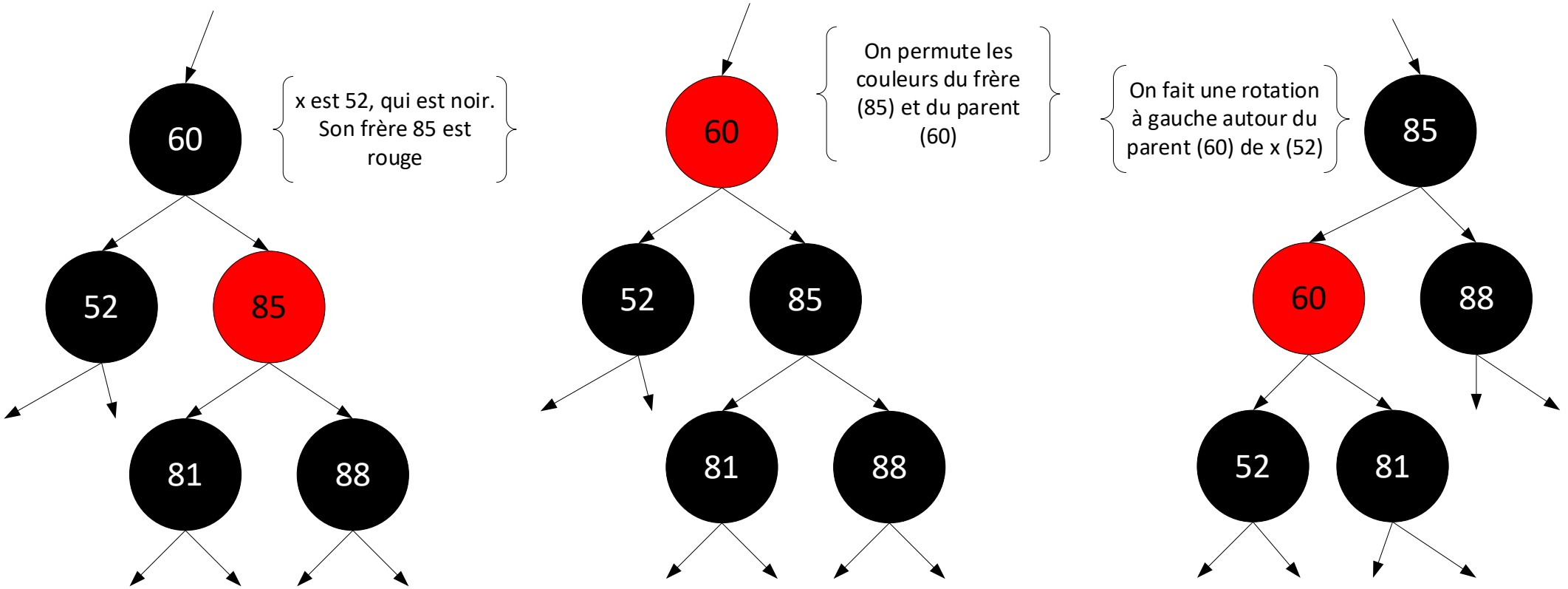
Suppression



Suppression

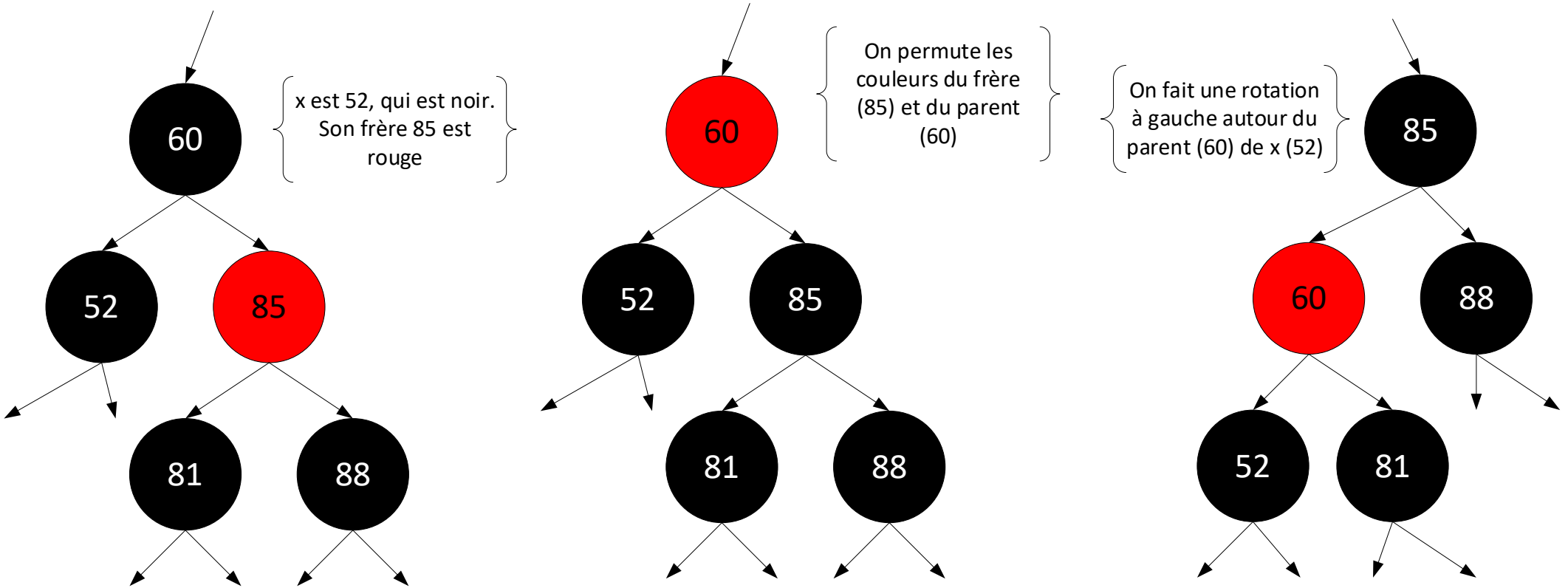


Suppression



Suppression

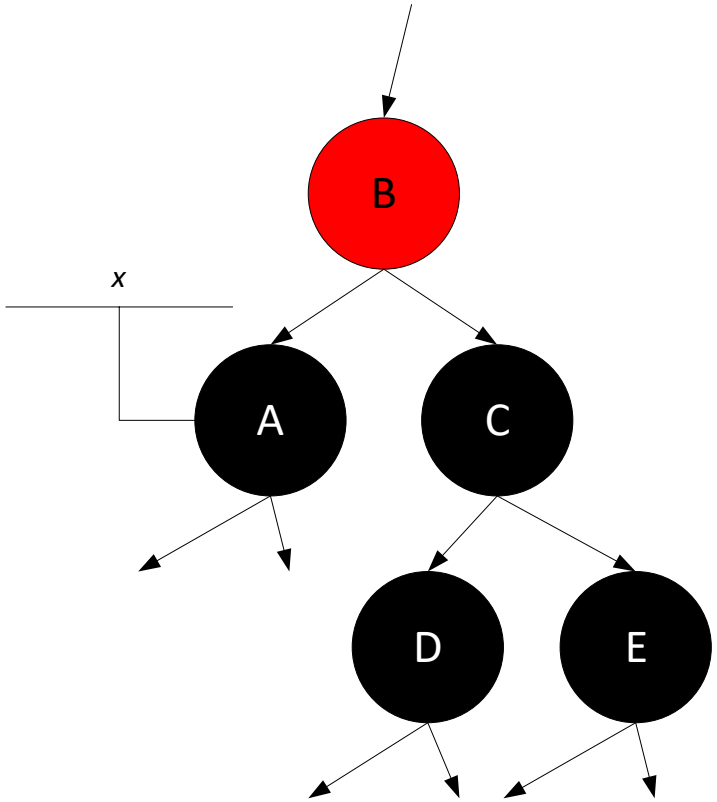
On n’a pas terminé (on tombe dans un des trois autres cas)



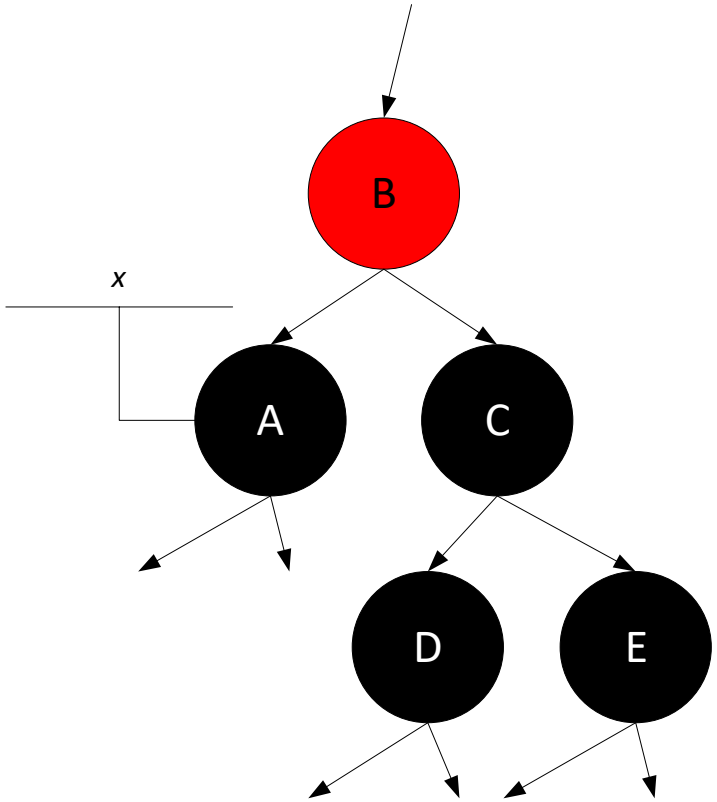
Suppression

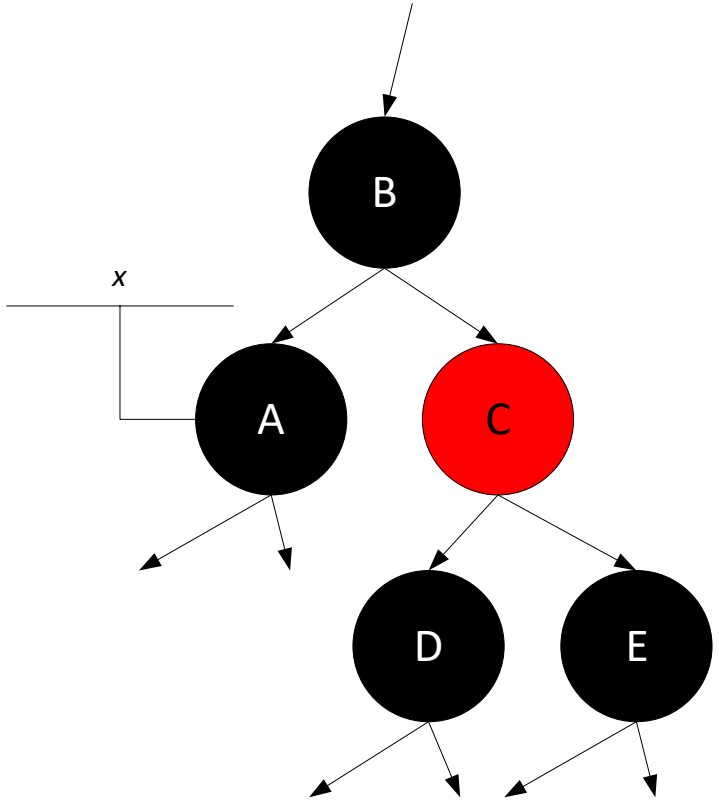
- Cas C.2) le frère de *x* est noir et les enfants du frère sont tous deux noirs
  - On sait que le parent de *x* peut être rouge ou noir
  - On recolorie le frère en rouge
  - Ensuite, si le parent est rouge, on le recolorie en noir et on a fini
  - Sinon…

Suppression



Suppression

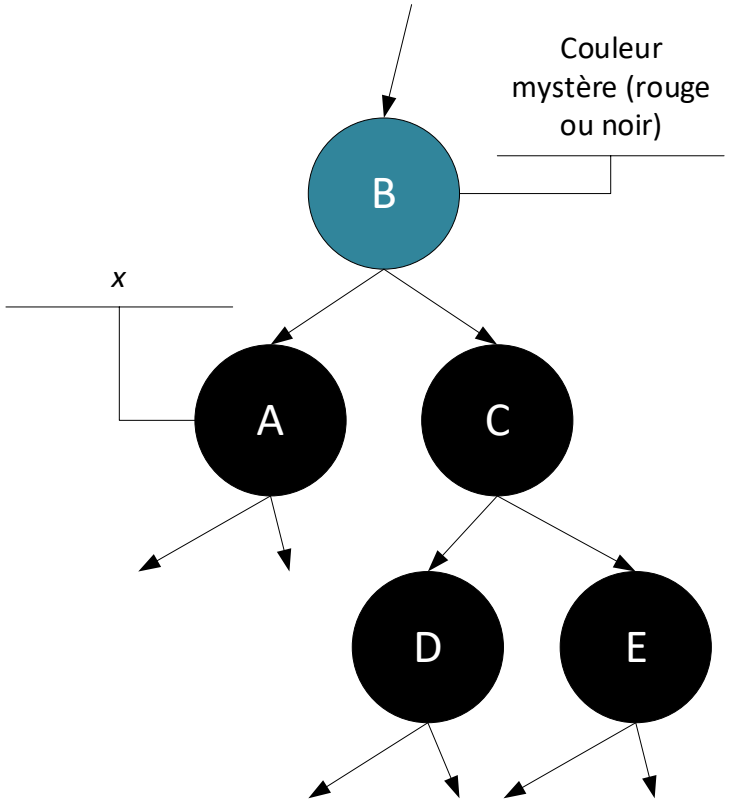




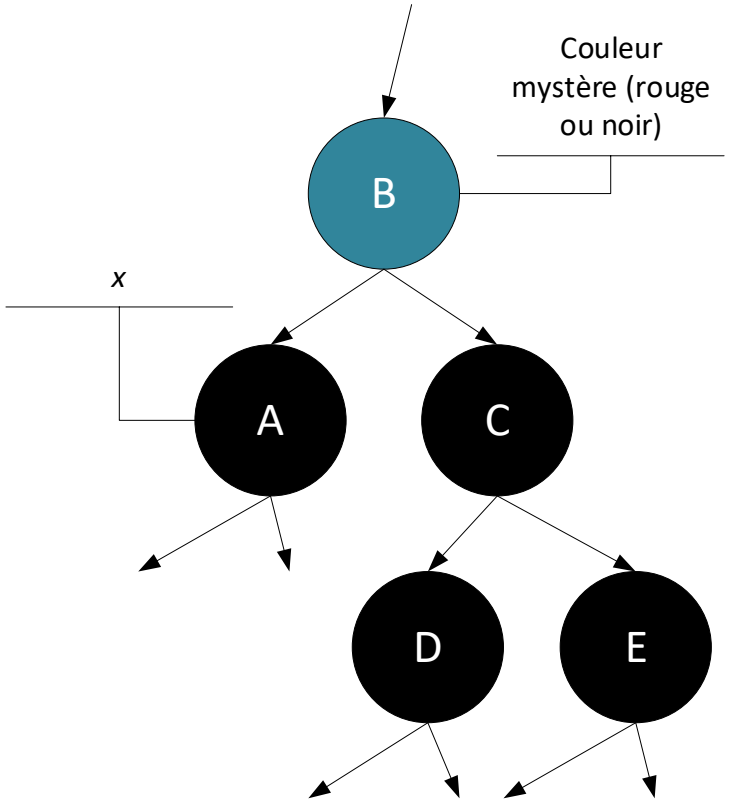
Suppression

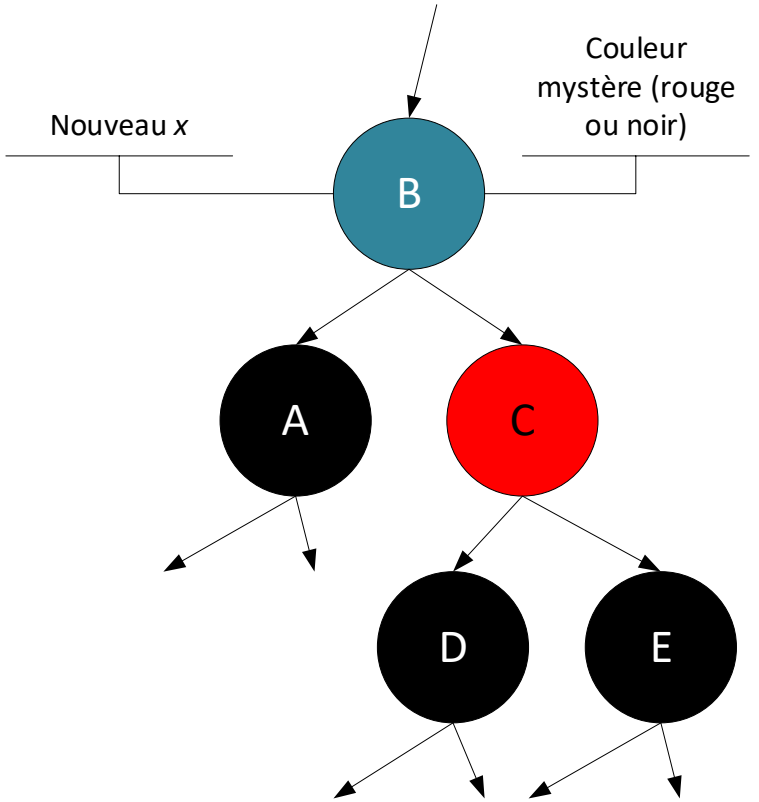
- Cas C.2) le frère de *x* est noir et les enfants du frère sont tous deux noirs
  - On sait que le parent de *x* peut être rouge ou noir
  - On recolorie le frère en rouge
  - Ensuite, si le parent est rouge, on le recolorie en noir et on a fini
  - Sinon … on considère que *x* est le parent de *x* et on reprend au cas C.1) en remontant vers la racine

Suppression



Suppression

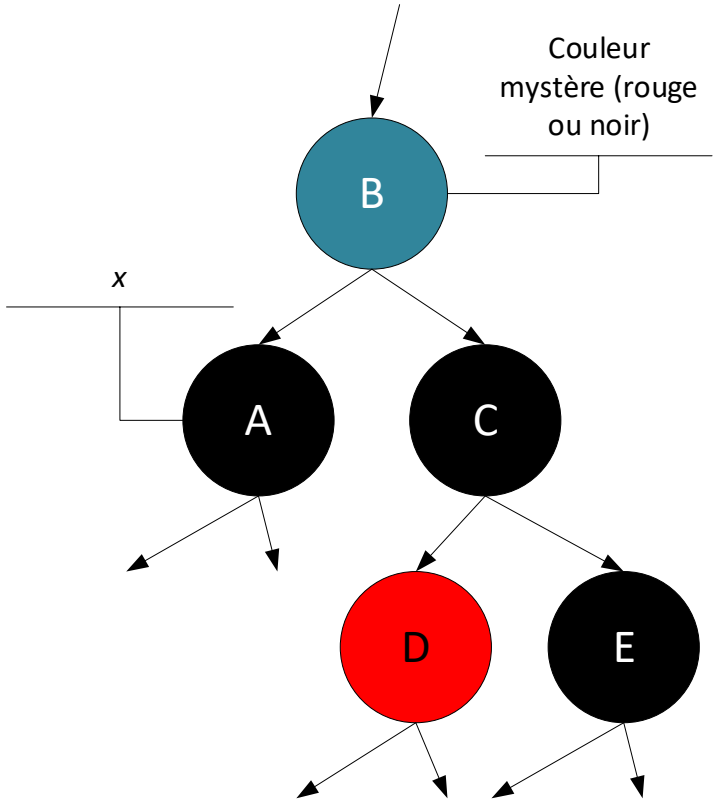




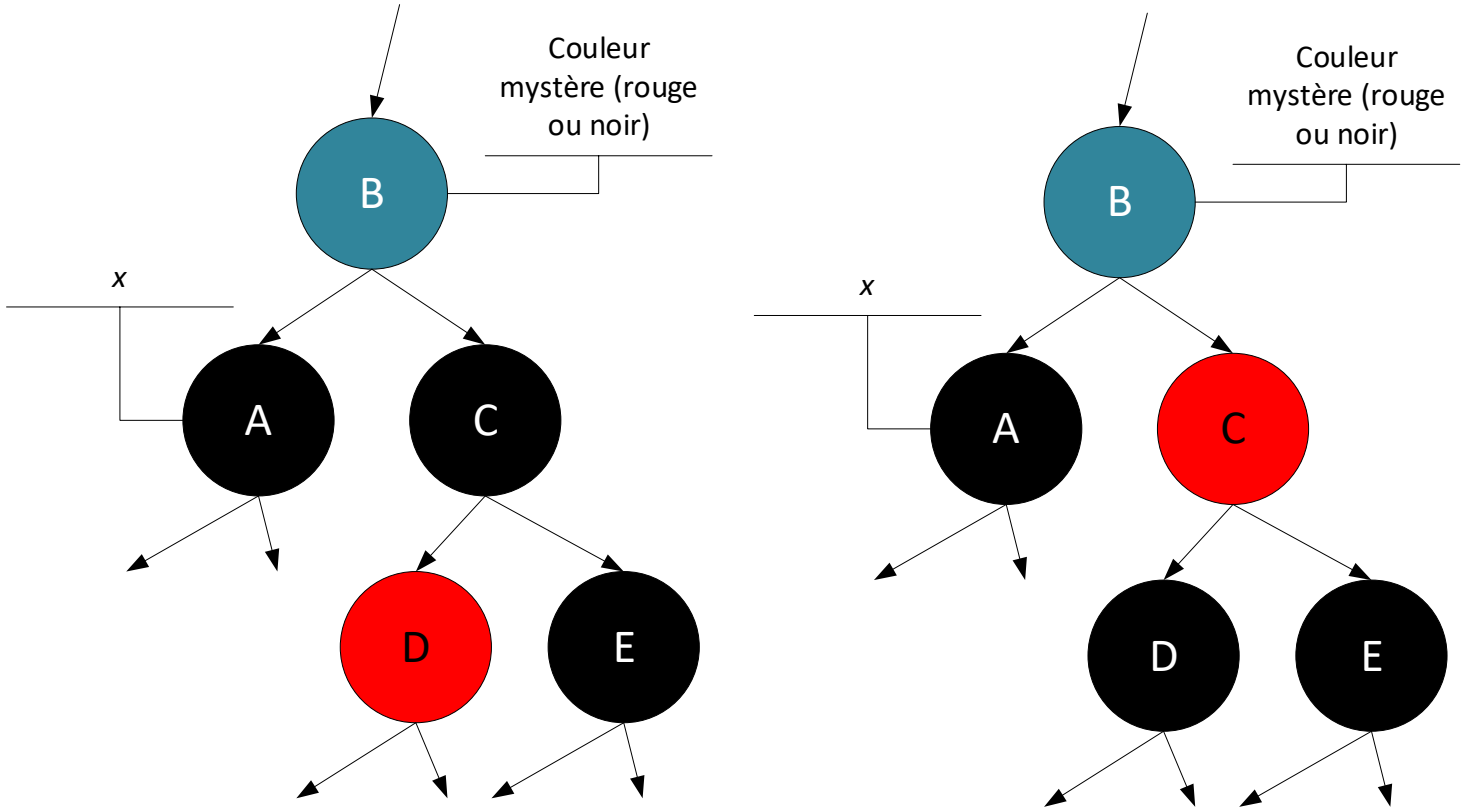
Suppression

- Cas C.3) le frère de *x* est noir, l'enfant de gauche du frère est rouge et l'enfant de droite du frère est noir
  - On permute les couleurs du frère et de son enfant de gauche
  - On fait une rotation à droite autour du frère
    - Cela amène au cas C.4)

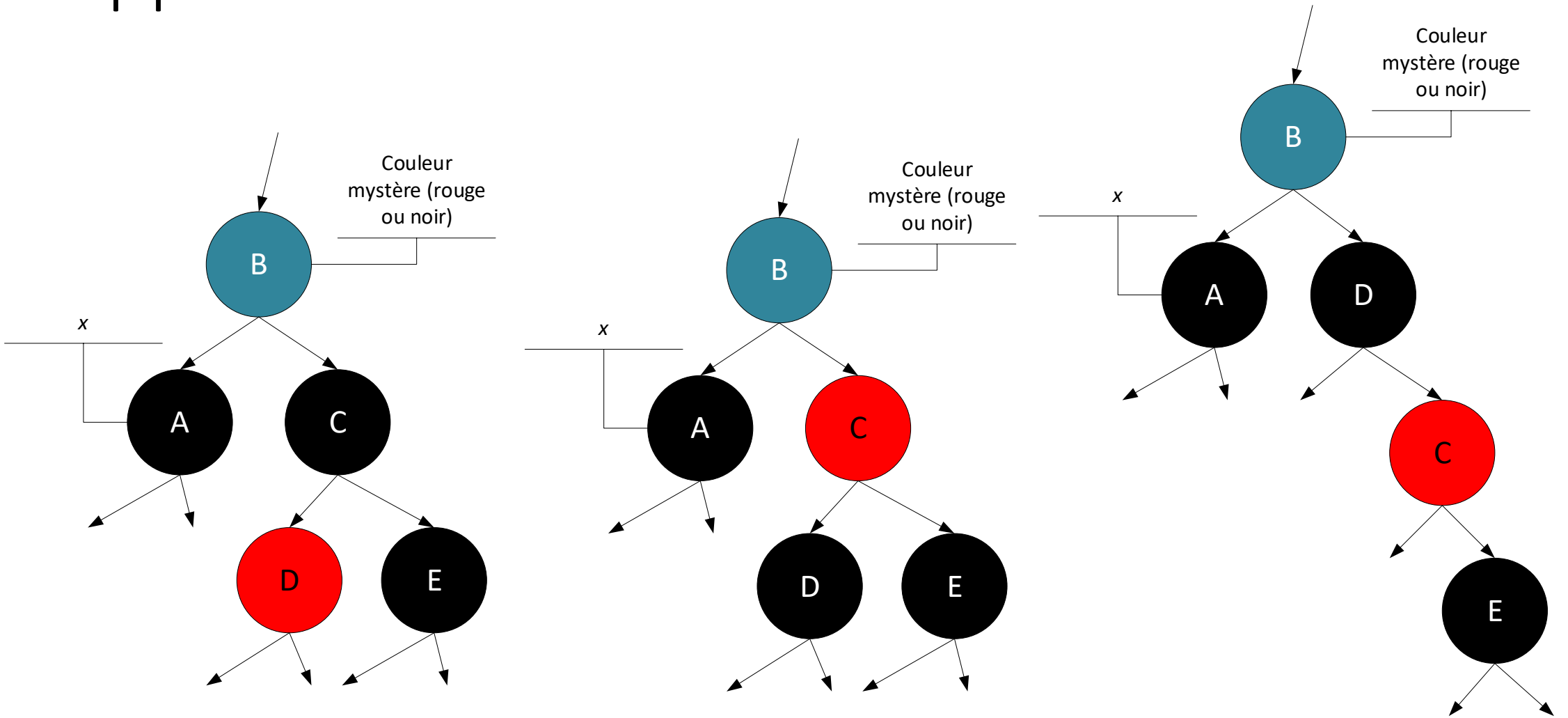
Suppression



Suppression

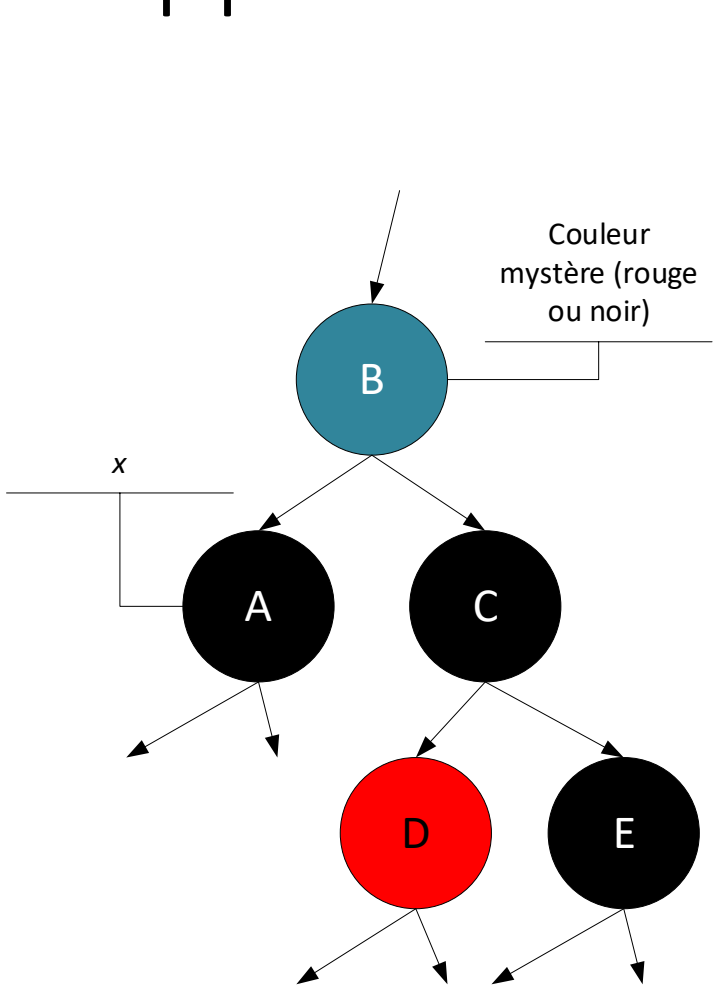


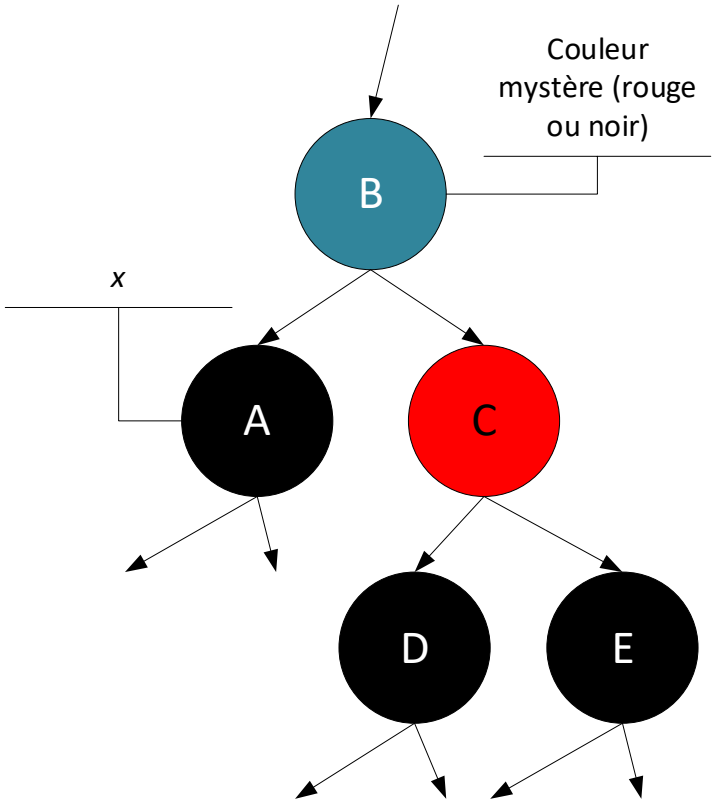
Suppression

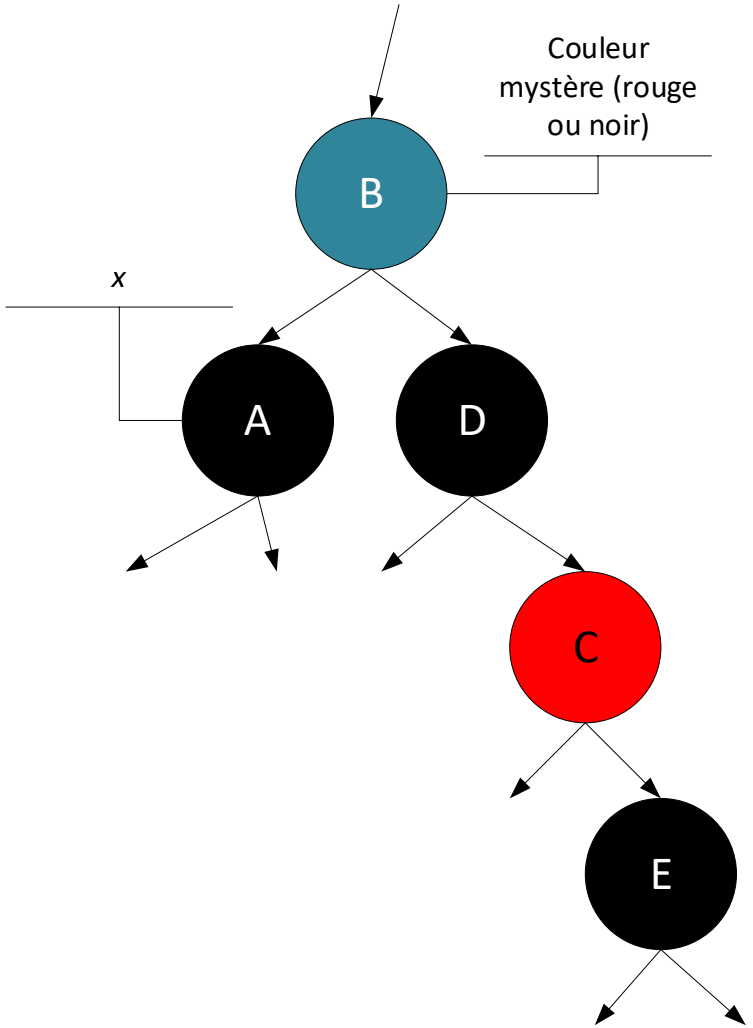


Suppression

On n’a pas terminé : on tombe dans le cas C.4)



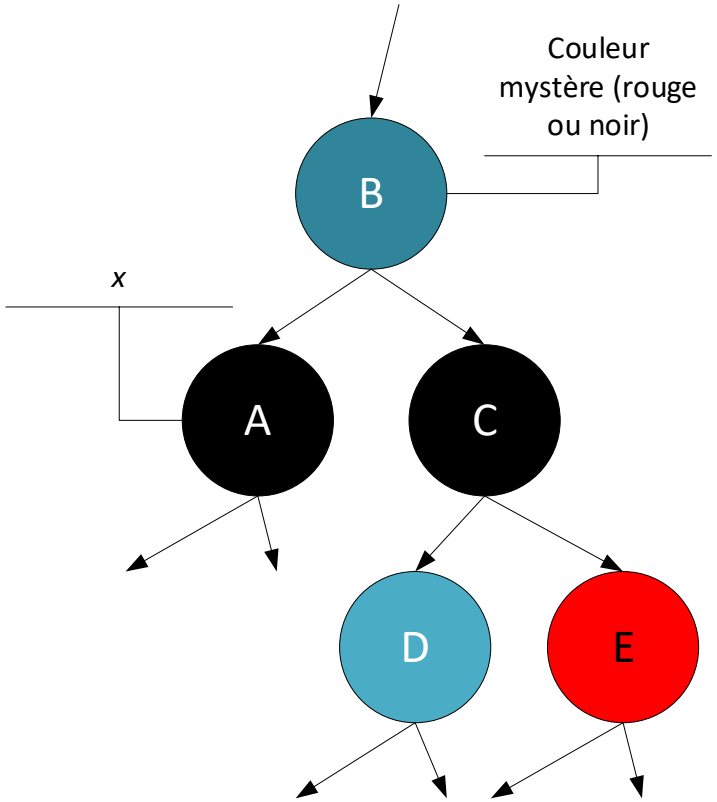




Suppression

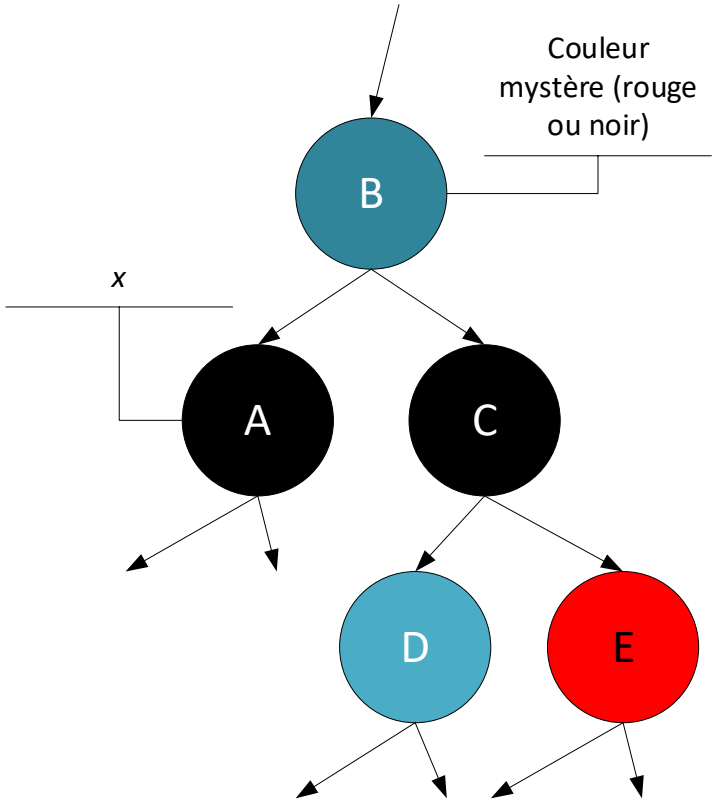
- Cas C.4) le frère de *x* est noir, l'enfant de gauche du frère est noir et l'enfant de droite du frère est rouge
  - C'est un cas terminal
  - On recolorie l'enfant de droite du frère en noir
  - On permute les couleurs du parent de *x* et du frère de *x*
  - On fait une rotation à gauche autour du parent de *x*

Suppression

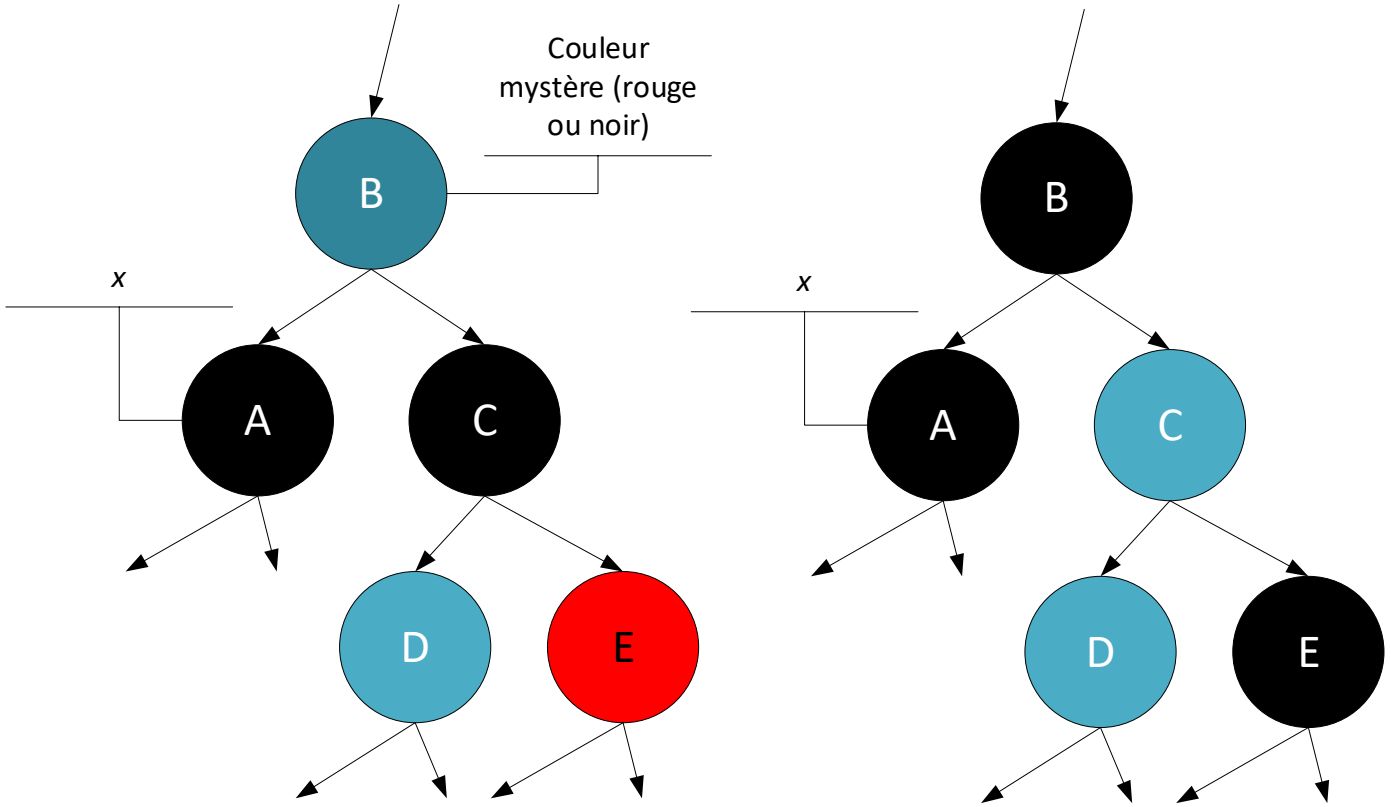


Suppression

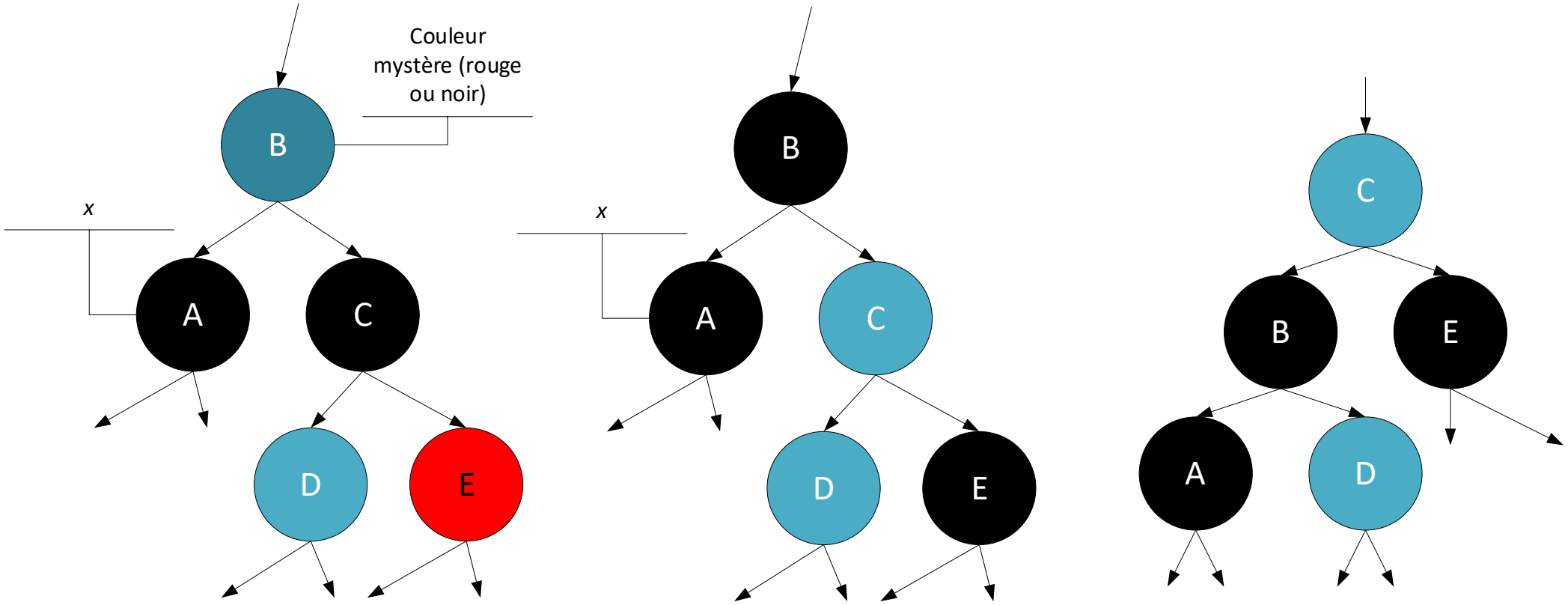
Que l'enfant de gauche du frère soit rouge ou noir n'importe pas ici



Suppression

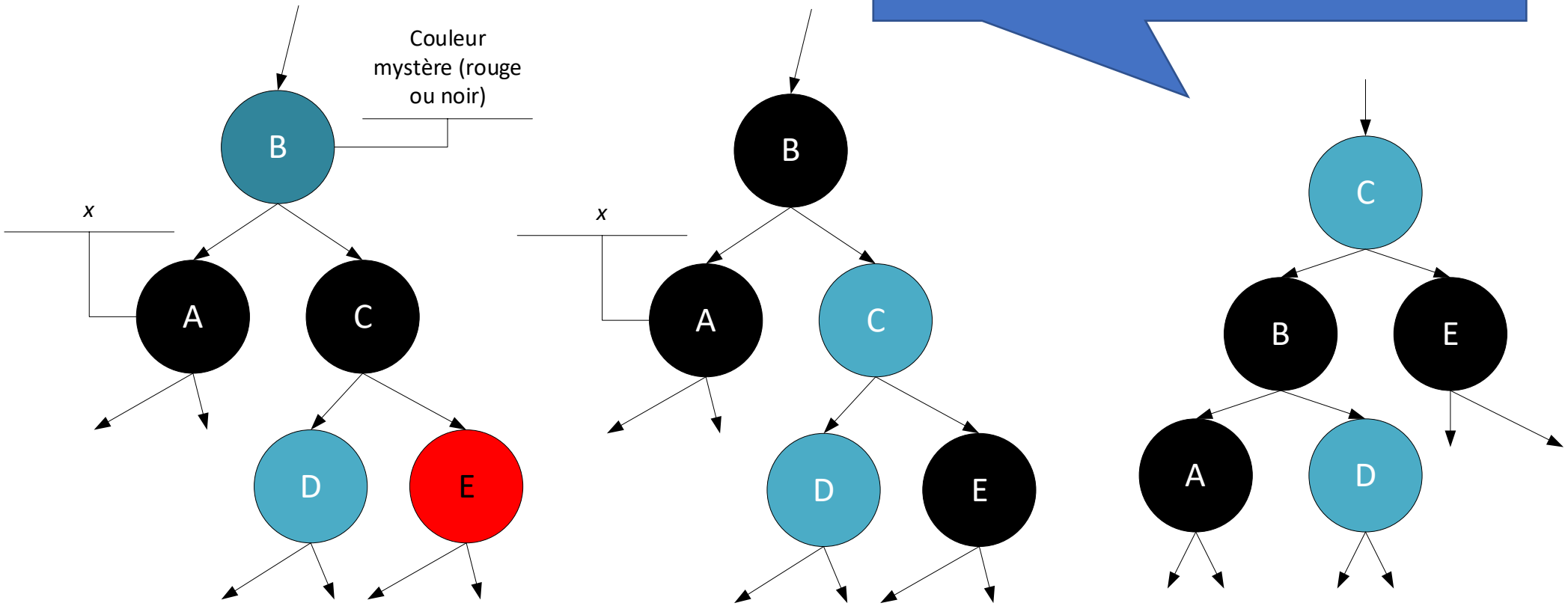


Suppression



Suppression

On a terminé : si la « racine » ici (nœud C) est la racine de l'arbre, alors on la colorie en noir, sinon on la laisse telle quelle



Suppression

- Pour les cas non-terminaux de suppression…
  - … les cas autres que A, B et C.4)
- … on poursuit ensuite l'effort en remontant vers la racine
  - … en n'oubliant pas qu'à la fin du processus, la racine doit être noire

Pourquoi un arbre bicolore?

Pourquoi un arbre bicolore?

- Rappel : l'enjeu est de modéliser un B-Tree d'ordre 3 ou 4
  - Les nœuds rouges (les liens rouges) sont des successions de valeurs
  - Les nœuds noirs (les liens noirs) sont des enfants

Pourquoi un arbre bicolore?

- L’intérêt d’un arbre bicolore est :
  - Qu’il est auto-équilibrant
    - … grâce au stratagème de coloration et aux algorithmes qui en profitent
    - …chaque insertion et chaque suppression amène des recolorations de nœuds

Pourquoi un arbre bicolore?

- L’intérêt d’un arbre bicolore est :
  - Qu’il est auto-équilibrant
  - La coloration peut se faire avec peu d’espace
    - Si on est astucieux, ça peut se limiter à un bit dans un pointeur
    - De manière plus « franche », le coût en espace est un booléen + alignement

Pourquoi un arbre bicolore?

- L’intérêt d’un arbre bicolore est :
  - Qu’il est auto-équilibrant
  - La coloration peut se faire avec peu d’espace
  - Les opérations d’insertion, de suppression et de recherche sont O(log n)

Pourquoi un arbre bicolore?

- L’intérêt d’un arbre bicolore est :
  - Qu’il est auto-équilibrant
  - La coloration peut se faire avec peu d’espace
  - Les opérations d’insertion, de suppression et de recherche sont O(log n)
  - Soit lgMax la longueur du chemin de la racine vers la feuille la plus éloignée
  - Soit lgMin la longueur du chemin de la racine vers la feuille la plus proche
  - Alors lgMax <=2*lgMin

Pourquoi un arbre bicolore?

- En comparaison avec un arbre AVL
  - Les recherches dans un arbre AVL sont légèrement plus rapides
    - Ils sont donc utilisés quand les quantités d'information à fouiller sont énormes

Pourquoi un arbre bicolore?

- En comparaison avec un arbre AVL
  - Les recherches dans un arbre AVL sont légèrement plus rapides
  - Le fait de maintenir une variation de hauteurs de moins de 2 entraîne des coûts de balancement plus importants dans un arbre AVL
    - L'arbre bicolore est moins strict
    - Ceci implique que les insertions et les suppressions dans un arbre bicolore sont en moyenne moins coûteuses que dans un arbre AVL

Pourquoi un arbre bicolore?

- En comparaison avec un arbre AVL
  - Les recherches dans un arbre AVL sont légèrement plus rapides
  - Le fait de maintenir une variation de hauteurs de moins de 2 entraîne des coûts de balancement plus importants dans un arbre AVL
  - Pour être efficace, un arbre AVL entreposera la balance ou la hauteur d'un nœud à même le nœud
    - Ceci est en général plus coûteux qu'un simple bit