Algorithme 2

Extension de structure de données

Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

Licence 2 Informatique – Semestre 4

Plan

Plan

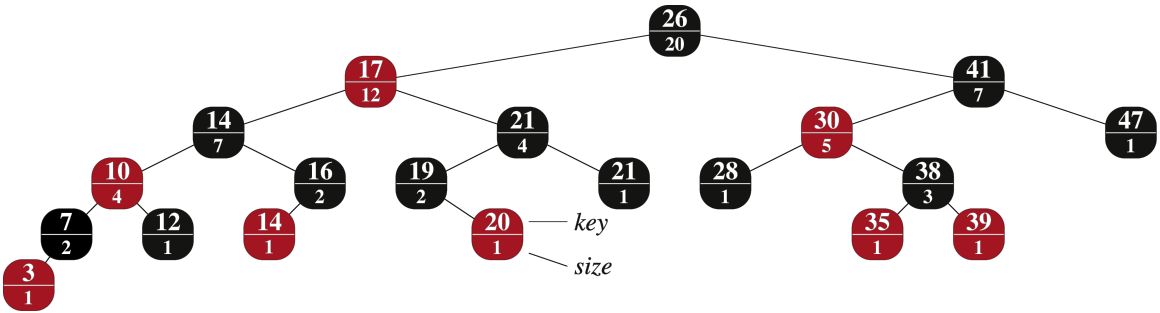
Contexte

- Le *rang* d'un élément dans un ensemble est sa position dans l'ensemble trié.
- Par exemple, le rang de 6 dans l'ensemble $\{4, 6, 1, 7\}$ est 3 : c'est le 3-ième plus petit élément.
- On cherche une structure de données efficace pour calculer le rang des éléments d'un ensemble qui change dynamiquement (insertions, suppressions).
- Plutôt que de développer une nouvelle structure, on s'appuie sur une structure existante : les arbres rouge-noir.
- On bénéficiera de l'efficacité de la structure pour les opérations basiques et on pourra calculer le rang en temps $O(\log n)$.

Extension

- On ajoute le champ *taille* : $x.taille$ est la taille du sous-arbre enraciné en x.
- Cette taille se calcule directement : $x.taille \leftarrow x.gauche.taille + x.droite.taille + 1$.
- La taille du nœud sentinelle est 0.

Exemple


26
20
17
12
41
7
14
7
21
4
30
5
47
1
10
4
16
2
19
2
21
1
28
1
38
3
7
2
12
1
14
1
20
1
35
1
39
1
3
1
key
size

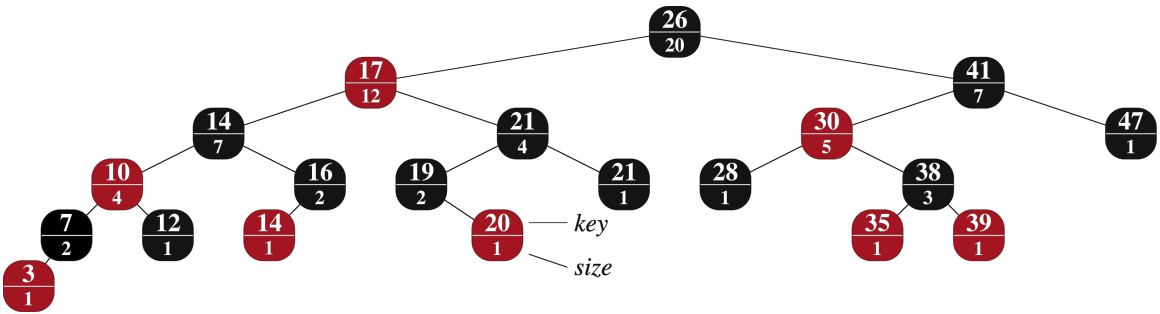
Pseudo-code de RÉCUPÉRER-RANG

RÉCUPÉRER-RANG(x, i)

$r \leftarrow x.gauche.taille + 1$
si $i = r$ **alors**
  retourner x
sinon si $i < r$ **alors**
  retourner RÉCUPÉRER-RANG$(x.gauche, i)$
sinon
  retourner RÉCUPÉRER-RANG$(x.droite, i - r)$

- `Récupérer-Rang` cherche l'élément de rang i dans le sous-arbre enraciné en x.
- r représente le rang de l'élément x dans le sous-arbre enraciné en x.
- Si le rang r est trop petit, on cherche l'élément de rang $i - r$ dans le sous-arbre de droite.
- La complexité en temps est $O(\log n)$.

Exemple

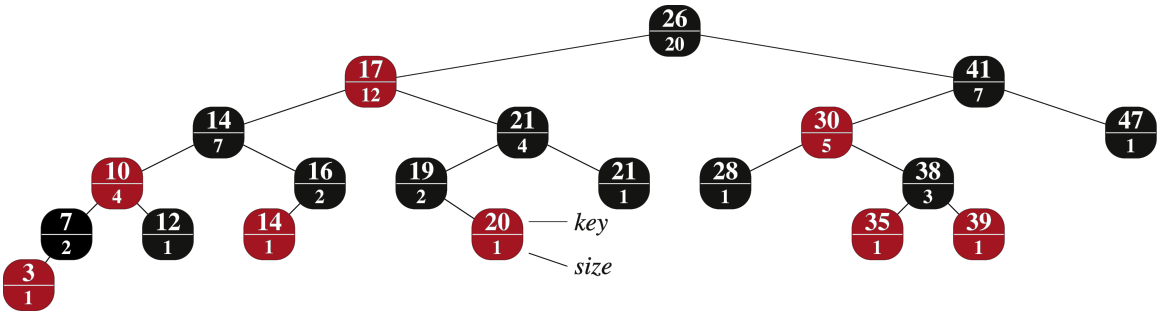

26
20
17
12
41
7
14
7
21
4
30
5
47
1
10
4
16
2
19
2
21
1
28
1
38
3
7
2
12
1
14
1
20
1
35
1
39
1
3
1
key
size

Pseudo-code de DÉTERMINER-RANG

```
DÉTERMINER-RANG(T, x)
r ← x.gauche.taille + 1
y ← x
tant que y ≠ T.racine faire
    si y = y.parent.droite alors
        r ← r + y.parent.gauche.taille + 1
    y ← y.parent
retourner r
```

- `Déterminer-Rang` détermine le rang de l'élément x.
- On remonte dans l'arbre de rangs en ajoutant tous les éléments inférieurs qui sont dans le sous-arbre du nœud voisin de gauche.
- La complexité en temps est $O(\log n)$.

Exemple


26
20
17
12
41
7
14
7
21
4
30
5
47
1
10
4
16
2
19
2
21
1
28
1
38
3
7
2
12
1
14
1
20
1
35
1
39
1
3
1
key
size

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle `tant que`, r est le rang de l'élément x dans le sous-arbre enracinée en y.

Initialisation À la première itération, r est le rang de x dans le sous-arbre enraciné en $x = y$.

Conservation On suppose l'invariant vrai pour un y donné. Il faut donc montrer que l'invariant est conservé pour $y.parent$.

- Si y est un enfant de gauche, r étant le rang de x dans y, il l'est aussi dans $y.parent$.
- Sinon, on ajoute 1 et la taille du sous-arbre de racine y à r et l'invariant est bien conservé.

Terminaison Lorsqu'on atteint la racine, le sous-arbre enraciné en y est en fait l'arbre de rangs T et l'invariant garantit que le rang r est celui de x dans T.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle `tant que`, r est le rang de l'élément x dans le sous-arbre enracinée en y.

Initialisation À la première itération, r est le rang de x dans le sous-arbre enraciné en $x = y$.

Conservation On suppose l'invariant vrai pour un y donné. Il faut donc montrer que l'invariant est conservé pour $y.parent$.

- Si y est un enfant de gauche, r étant le rang de x dans y, il l'est aussi dans $y.parent$.
- Sinon, on ajoute 1 et la taille du sous-arbre du nœud voisin au rang r et l'invariant est bien conservé.

Terminaison Lorsqu'on atteint la racine, le sous-arbre enraciné en y est en fait l'arbre de rangs T et l'invariant garantit que le rang r est celui de x dans T.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle `tant que`, r est le rang de l'élément x dans le sous-arbre enracinée en y.

Initialisation À la première itération, r est le rang de x dans le sous-arbre enraciné en $x = y$.

Conservation On suppose l'invariant vrai pour un y donné. Il faut donc montrer que l'invariant est conservé pour $y.parent$.

- Si y est un enfant de gauche, r étant le rang de x dans y, il l'est aussi dans $y.parent$.
- Sinon, on ajoute 1 et la taille du sous-arbre du nœud voisin au rang r et l'invariant est bien conservé.

Terminaison Lorsqu'on atteint la racine, le sous-arbre enraciné en y est en fait l'arbre de rangs T et l'invariant garantit que le rang r est celui de x dans T.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle `tant que`, r est le rang de l'élément x dans le sous-arbre enracinée en y.

Initialisation À la première itération, r est le rang de x dans le sous-arbre enraciné en $x = y$.

Conservation On suppose l'invariant vrai pour un y donné. Il faut donc montrer que l'invariant est conservé pour $y.parent$.

- Si y est un enfant de gauche, r étant le rang de x dans y, il l'est aussi dans $y.parent$.
- Sinon, on ajoute 1 et la taille du sous-arbre du nœud voisin au rang r et l'invariant est bien conservé.

Terminaison Lorsqu'on atteint la racine, le sous-arbre enraciné en y est en fait l'arbre de rangs T et l'invariant garantit que le rang r est celui de x dans T.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle `tant que`, r est le rang de l'élément x dans le sous-arbre enracinée en y.

Initialisation À la première itération, r est le rang de x dans le sous-arbre enraciné en $x = y$.

Conservation On suppose l'invariant vrai pour un y donné. Il faut donc montrer que l'invariant est conservé pour $y.parent$.

- Si y est un enfant de gauche, r étant le rang de x dans y, il l'est aussi dans $y.parent$.
- Sinon, on ajoute 1 et la taille du sous-arbre du nœud voisin au rang r et l'invariant est bien conservé.

Terminaison Lorsqu'on atteint la racine, le sous-arbre enraciné en y est en fait l'arbre de rangs T et l'invariant garantit que le rang r est celui de x dans T.

Preuve de validité

Invariant de boucle Au début de chaque itération de la boucle `tant que`, r est le rang de l'élément x dans le sous-arbre enracinée en y.

Initialisation À la première itération, r est le rang de x dans le sous-arbre enraciné en $x = y$.

Conservation On suppose l'invariant vrai pour un y donné. Il faut donc montrer que l'invariant est conservé pour $y.parent$.

- Si y est un enfant de gauche, r étant le rang de x dans y, il l'est aussi dans $y.parent$.
- Sinon, on ajoute 1 et la taille du sous-arbre du nœud voisin au rang r et l'invariant est bien conservé.

Terminaison Lorsqu'on atteint la racine, le sous-arbre enraciné en y est en fait l'arbre de rangs T et l'invariant garantit que le rang r est celui de x dans T.

Conserver la taille des sous-arbres

- L'ajout d'un champ `taille` permet de déterminer le rang efficacement sur un arbre rouge-noir déjà construit.
- Quel est l'impact sur les autres opérations pour cette structure ?
- Les requêtes d'interrogations ne sont pas impactées car elles ignorent ce champ.
- Les opérations de modifications (insertions et suppressions) doivent être modifiées à coût constant.
- Pour l'insertion (et la suppression), l'ajout initial d'un nœud en tant que feuille et la mise à jour des couleurs lors de la correction ne sont pas impactées.
- Seules les rotations ont besoin d'être adaptées pour gérer le champ `taille`.

Conserver la taille des sous-arbres

- L'ajout d'un champ `taille` permet de déterminer le rang efficacement sur un arbre rouge-noir déjà construit.
- Quel est l'impact sur les autres opérations pour cette structure ?
- Les requêtes d'interrogations ne sont pas impactées car elles ignorent ce champ.
- Les opérations de modifications (insertions et suppressions) doivent être modifiées à coût constant.
- Pour l'insertion (et la suppression), l'ajout initial d'un nœud en tant que feuille et la mise à jour des couleurs lors de la correction ne sont pas impactées.
- Seules les rotations ont besoin d'être adaptées pour gérer le champ `taille`.

Conserver la taille des sous-arbres

- L'ajout d'un champ `taille` permet de déterminer le rang efficacement sur un arbre rouge-noir déjà construit.
- Quel est l'impact sur les autres opérations pour cette structure ?
- Les requêtes d'interrogations ne sont pas impactées car elles ignorent ce champ.
- Les opérations de modifications (insertions et suppressions) doivent être modifiées à coût constant.
- Pour l'insertion (et la suppression), l'ajout initial d'un nœud en tant que feuille et la mise à jour des couleurs lors de la correction ne sont pas impactées.
- Seules les rotations ont besoin d'être adaptées pour gérer le champ `taille`.

Conserver la taille des sous-arbres

- L'ajout d'un champ `taille` permet de déterminer le rang efficacement sur un arbre rouge-noir déjà construit.
- Quel est l'impact sur les autres opérations pour cette structure ?
- Les requêtes d'interrogations ne sont pas impactées car elles ignorent ce champ.
- Les opérations de modifications (insertions et suppressions) doivent être modifiées à coût constant.
- Pour l'insertion (et la suppression), l'ajout initial d'un nœud en tant que feuille et la mise à jour des couleurs lors de la correction ne sont pas impactées.
- Seules les rotations ont besoin d'être adaptées pour gérer le champ `taille`.

Conserver la taille des sous-arbres

- L'ajout d'un champ `taille` permet de déterminer le rang efficacement sur un arbre rouge-noir déjà construit.
- Quel est l'impact sur les autres opérations pour cette structure ?
- Les requêtes d'interrogations ne sont pas impactées car elles ignorent ce champ.
- Les opérations de modifications (insertions et suppressions) doivent être modifiées à coût constant.
- Pour l'insertion (et la suppression), l'ajout initial d'un nœud en tant que feuille et la mise à jour des couleurs lors de la correction ne sont pas impactées.
- Seules les rotations ont besoin d'être adaptées pour gérer le champ `taille`.

Conserver la taille des sous-arbres

- L'ajout d'un champ `taille` permet de déterminer le rang efficacement sur un arbre rouge-noir déjà construit.
- Quel est l'impact sur les autres opérations pour cette structure ?
- Les requêtes d'interrogations ne sont pas impactées car elles ignorent ce champ.
- Les opérations de modifications (insertions et suppressions) doivent être modifiées à coût constant.
- Pour l'insertion (et la suppression), l'ajout initial d'un nœud en tant que feuille et la mise à jour des couleurs lors de la correction ne sont pas impactées.
- Seules les rotations ont besoin d'être adaptées pour gérer le champ `taille`.

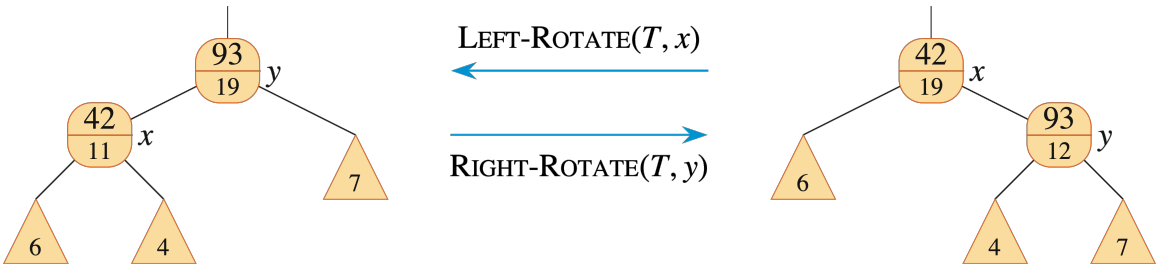
Rotations

ROTATION-GAUCHE(T, x)

...

$y.taile \leftarrow x.taille$

$x.taille \leftarrow x.gauche.taille + x.droite.taille + 1$



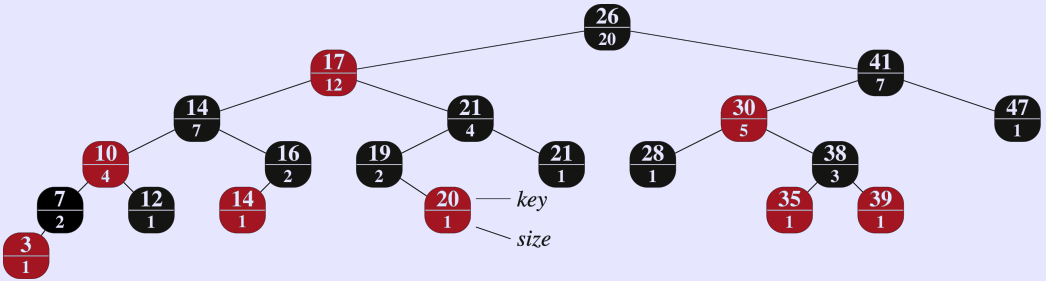
Récapitulatif de l'extension

L'extension d'une structure de données peut être divisée en quatre étapes :

- choisir une structure de données sous-jacente ;
- déterminer les informations supplémentaires à gérer dans la structure ;
- vérifier que les informations supplémentaires seront compatibles avec les opérations habituelles de modification de la structure ;
- créer de nouvelles opérations.

Question

On applique DÉTERMINER-RANG sur cet arbre de rangs pour les nœuds 14, 21, 35 et 47.

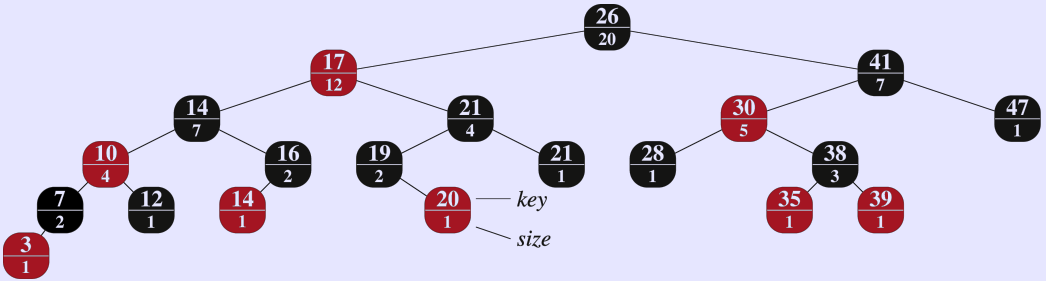


Quelle séquence d'opérations n'est jamais réalisée ?

1. $1 + 4$
2. $1 + 2 + 1 + 7 + 1$
3. $1 + 2 + 12 + 1$
4. $1 + 5 + 1 + 12 + 1$

Question

On applique DÉTERMINER-RANG sur cet arbre de rangs pour les nœuds 14, 21, 35 et 47.



Quelle séquence d'opérations n'est jamais réalisée ?

1. $1 + 4$ (14)
2. $1 + 2 + 1 + 7 + 1$ (21)
3. $1 + 2 + 12 + 1$ ✓$(1 + 1 + 1 + 12 + 1)$
4. $1 + 5 + 1 + 12 + 1$ (47)

Correction

- $1 + 2 + 12 + 1 = 16$: on trouve que le 16-ième élément est 35.
- Pour 35, on a $1(35) + 1(30.\mathit{gauche.taille}) + 1(30) + 12(26.\mathit{gauche.taille}) + 1(26)$.

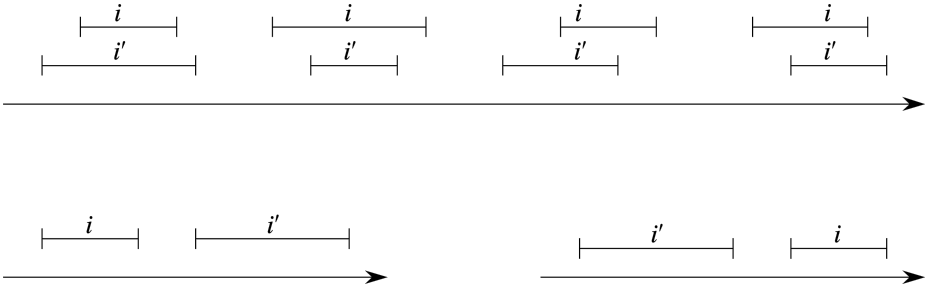
Plan

Contexte

- On considère des événements qui occupent chacune une période de temps avec un début et une fin et on souhaite trouver tous les événements se produisant pendant un intervalle donné (ou au moins un).
- En 2 dimensions, le problème consiste à trouver l'ensemble des éléments géographiques situés dans un rectangle de recherche.
- En 3 dimensions, il s'agit de trouver tous les éléments visibles.
- On va s'intéresser à une structure de données efficace pour le cas à 1 dimension.

Intervalle

- Un *intervalle* i consiste en un début, $i.début$, et une fin, $i.fin$.
- Pour deux intervalles i et i', soit ils se recoupent, soit l'un est avant l'autre.



Curiosité

La *trichomie d'intervalle* est le nom de la propriété qui fait que soit i et i' se recoupent, soit i précède i', soit i' précède i.

Opérations

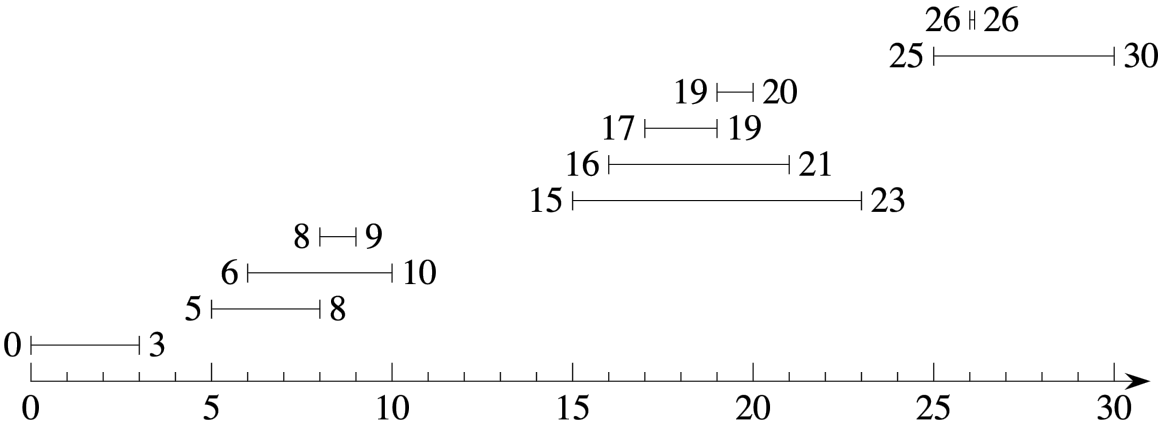
- Un *arbre d'intervalles* (1980) aura 3 opérations :

  RECHERCHER-INTERVALLE retourne un intervalle qui se recoupe avec celui recherché ;
  INSÉRER-INTERVALLE ajoute un intervalle dans l'arbre ;
  SUPPRIMER-INTERVALLE supprime un intervalle de l'arbre.
- On souhaite une complexité en temps en $O(\log n)$ pour chacune de ces requêtes.

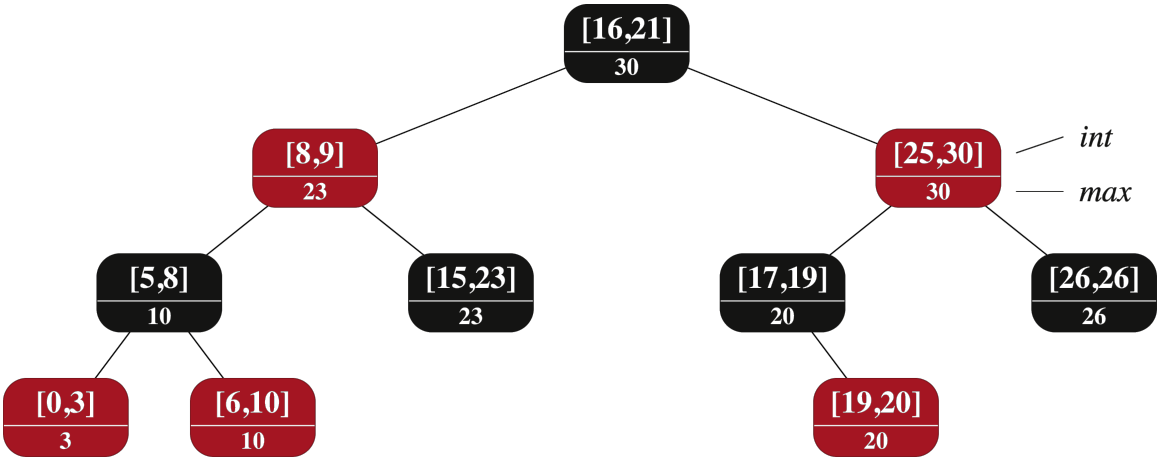
Extension d'une structure de données

1. Choix de la structure de données sous-jacente : arbre rouge-noir où chaque nœud x possède un champ $x.int$ pour l'intervalle. La clé sera $x.int.début$.
2. Informations supplémentaires : on ajoute un champ $x.max$ qui représente la valeur maximale des $x.int.fin$ dans le sous-arbre enraciné en x.
3. Compatibilité avec l'insertion et la suppression : lors d'une rotation, on peut recalculer $x.max \leftarrow \max(x.int.fin, x.gauche.max, x.droite.max)$ en temps $\Theta(1)$.
4. Nouvelle opération à concevoir : rechercher (c'est ce qu'il reste à faire).

Exemple



Exemple


[16,21]
30
[8,9]
23
[25,30]
30
int
max
[5,8]
10
[15,23]
23
[17,19]
20
[26,26]
26
[0,3]
3
[6,10]
10
[19,20]
20

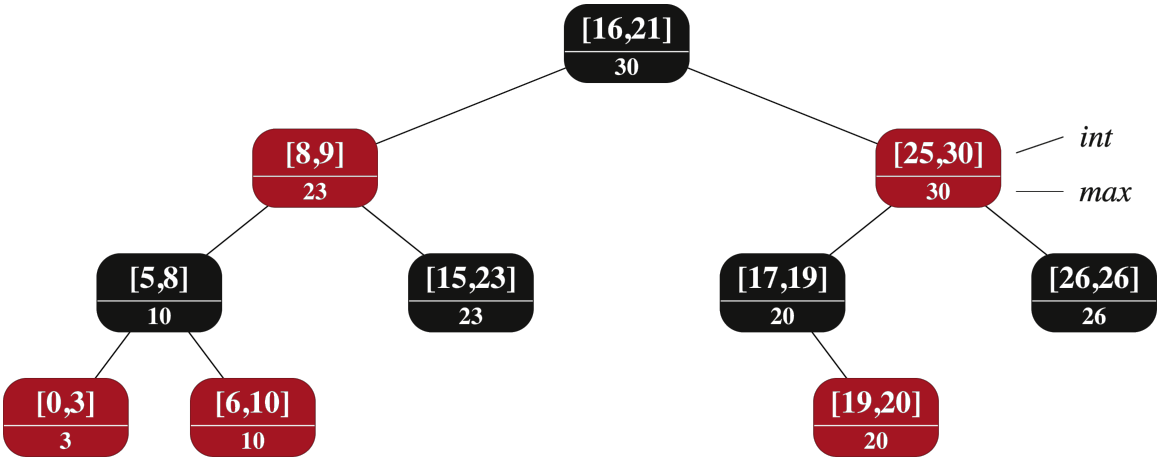
Pseudo-code de RECHERCHER-INTERVALLE

RECHERCHER-INTERVALLE(T, i)

$x \leftarrow T.racine$
tant que $x \neq T.nil$ **et** i ne recoupe pas $x.int$ **faire**
  si $x.gauche \neq T.nil$ **et** $i.début \leq x.gauche.max$ **alors**
    $x \leftarrow x.gauche$
  sinon
    $x \leftarrow x.droite$
retourner x

La complexité en temps est $O(\log n)$.

Exemple : $[22, 26]$ et $[11, 14]$



Preuve de validité (invariant)

Invariant de boucle Au début de chaque itération de la boucle `tant que`, si l'arbre T contient un intervalle qui recoupe i, alors il existe un tel intervalle dans le sous-arbre enraciné en x.

Initialisation À la première itération, x pointe vers la racine de l'arbre et l'invariant est direct.

Conservation On suppose l'invariant vrai pour un x donné. Il faut donc montrer que l'invariant est conservé pour $x.gauche$ ou $x.droite$.

- Pour $x.gauche$, s'il n'y a pas d'intervalle se recoupant avec i dans $x.gauche$, il n'y en a pas non plus dans $x.droite$ (ce qui restera à prouver).
- Pour $x.droite$, soit $x.gauche$ est NIL, soit l'intervalle i commence après la fin de tous les intervalles de $x.gauche$ ($i.debut > x.gauche.max$).

Terminaison Si x est NIL, le sous-arbre enraciné en x est vide et il n'y a donc pas d'intervalle se recoupant avec i. Sinon, $x.int$ se recoupe avec i.

Preuve de validité (invariant)

Invariant de boucle Au début de chaque itération de la boucle `tant que`, si l'arbre T contient un intervalle qui recoupe i, alors il existe un tel intervalle dans le sous-arbre enraciné en x.

Initialisation À la première itération, x pointe vers la racine de l'arbre et l'invariant est direct.

Conservation On suppose l'invariant vrai pour un x donné. Il faut donc montrer que l'invariant est conservé pour $x.gauche$ ou $x.droite$.

- Pour $x.gauche$, s'il n'y a pas d'intervalle se recoupant avec i dans $x.gauche$, il n'y en a pas non plus dans $x.droite$ (c'est ce qui resterait à prouver).
- Pour $x.droite$, soit $x.gauche$ est NIL, soit l'intervalle i commence après la fin de tous les intervalles de gauche ($i.début > x.gauche.max$).

Terminaison Si x est NIL, le sous-arbre enraciné en x est vide et il n'y a donc pas d'intervalle se recoupant avec i. Sinon, $x.int$ se recoupe avec i.

Preuve de validité (invariant)

Invariant de boucle Au début de chaque itération de la boucle `tant que`, si l'arbre T contient un intervalle qui recoupe i, alors il existe un tel intervalle dans le sous-arbre enraciné en x.

Initialisation À la première itération, x pointe vers la racine de l'arbre et l'invariant est direct.

Conservation On suppose l'invariant vrai pour un x donné. Il faut donc montrer que l'invariant est conservé pour $x.gauche$ ou $x.droite$.

- Pour $x.gauche$, s'il n'y a pas d'intervalle se recoupant avec i dans $x.gauche$, il n'y en a pas non plus dans $x.droite$ (c'est ce qui resterait à prouver).
- Pour $x.droite$, soit $x.gauche$ est NIL, soit l'intervalle i commence après la fin de tous les intervalles de gauche ($i.début \leq x.gauche.max$).

Terminaison Si x est NIL, le sous-arbre enraciné en x est vide et il n'y a donc pas d'intervalle se recoupant avec i. Sinon, $x.int$ se recoupe avec i.

Preuve de validité (invariant)

Invariant de boucle Au début de chaque itération de la boucle `tant que`, si l'arbre T contient un intervalle qui recoupe i, alors il existe un tel intervalle dans le sous-arbre enraciné en x.

Initialisation À la première itération, x pointe vers la racine de l'arbre et l'invariant est direct.

Conservation On suppose l'invariant vrai pour un x donné. Il faut donc montrer que l'invariant est conservé pour $x.gauche$ ou $x.droite$.

- Pour $x.gauche$, s'il n'y a pas d'intervalle se recoupant avec i dans $x.gauche$, il n'y en a pas non plus dans $x.droite$ (c'est ce qui resterait à prouver).
- Pour $x.droite$, soit $x.gauche$ est NIL, soit l'intervalle i commence après la fin de tous les intervalles de gauche ($i.début \leq x.gauche.max$).

Terminaison Si x est NIL, le sous-arbre enraciné en x est vide et il n'y a donc pas d'intervalle se recoupant avec i. Sinon, $x.int$ se recoupe avec i.

Preuve de validité (invariant)

Invariant de boucle Au début de chaque itération de la boucle `tant que`, si l'arbre T contient un intervalle qui recoupe i, alors il existe un tel intervalle dans le sous-arbre enraciné en x.

Initialisation À la première itération, x pointe vers la racine de l'arbre et l'invariant est direct.

Conservation On suppose l'invariant vrai pour un x donné. Il faut donc montrer que l'invariant est conservé pour $x.gauche$ ou $x.droite$.

- Pour $x.gauche$, s'il n'y a pas d'intervalle se recoupant avec i dans $x.gauche$, il n'y en a pas non plus dans $x.droite$ (c'est ce qui resterait à prouver).
- Pour $x.droite$, soit $x.gauche$ est *NIL*, soit l'intervalle i commence après la fin de tous les intervalles de gauche ($i.début \leq x.gauche.max$).

Terminaison Si x est *NIL*, le sous-arbre enraciné en x est vide et il n'y a donc pas d'intervalle se recoupant avec i. Sinon, $x.int$ se recoupe avec i.

Preuve de validité (invariant)

Invariant de boucle Au début de chaque itération de la boucle `tant que`, si l'arbre T contient un intervalle qui recoupe i, alors il existe un tel intervalle dans le sous-arbre enraciné en x.

Initialisation À la première itération, x pointe vers la racine de l'arbre et l'invariant est direct.

Conservation On suppose l'invariant vrai pour un x donné. Il faut donc montrer que l'invariant est conservé pour $x.gauche$ ou $x.droite$.

- Pour $x.gauche$, s'il n'y a pas d'intervalle se recoupant avec i dans $x.gauche$, il n'y en a pas non plus dans $x.droite$ (c'est ce qui resterait à prouver).
- Pour $x.droite$, soit $x.gauche$ est NIL, soit l'intervalle i commence après la fin de tous les intervalles de gauche ($i.début \leq x.gauche.max$).

Terminaison Si x est NIL, le sous-arbre enraciné en x est vide et il n'y a donc pas d'intervalle se recoupant avec i. Sinon, $x.int$ se recoupe avec i.

Question

On ajoute les intervalles $[4, 5]$, $[2, 12]$, $[6, 7]$, $[1, 10]$, $[3, 11]$, $[5, 5]$ et $[9, 13]$ dans cet ordre dans un arbre d'intervalles initialement vide. Que retourne la procédure RECHERCHER-INTERVALLE pour $i = [8, 10]$?

1. $[2, 12]$
2. $[1, 10]$
3. $[3, 11]$
4. $[9, 13]$

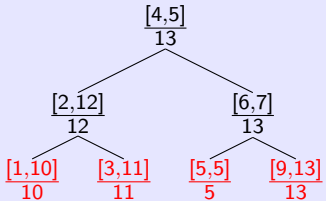
Question

On ajoute les intervalles $[4, 5]$, $[2, 12]$, $[6, 7]$, $[1, 10]$, $[3, 11]$, $[5, 5]$ et $[9, 13]$ dans cet ordre dans un arbre d'intervalles initialement vide. Que retourne la procédure RECHERCHER-INTERVALLE pour $i = [8, 10]$?

1. $[2, 12]$ ✓
2. $[1, 10]$
3. $[3, 11]$
4. $[9, 13]$

Correction

On obtient l'arbre rouge-noir :



Comme $8 \leq 12$, on part à gauche et $[2, 12]$ se recoupe avec $[8, 10]$.

Plan

Récapitulatif sur les structures de données

Structure	trouver [1]	succ./pred. [2]	min./max.	insert.	suppr. [2]
Tableau non ordonné	$O(n)$	$O(n)$	$O(n)$	$\Theta(1)^{\text{am}}$	$\Theta(1)$
Tableau ordonné	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(n)$	$\Theta(1)$	$\Theta(1)$
Table de hachage	$\Theta(1)^{\text{moy}}$	$O(n)$	$O(n)$	$\Theta(1)^{\text{moy}}$	$\Theta(1)^{\text{moy}}$
Tas (binaire)	$O(n)$	$O(n)$	$O(n)/\Theta(1)$	$O(\log n)$	$O(\log n)$
Tas de Fibonacci	$O(n)$	$O(n)$	$O(n)/\Theta(1)$	$\Theta(1)$	$O(\log n)^{\text{am}}$
Arbre rech. non équilibré	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Arbre rech. équilibré	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

1. La rechercher détermine la présence d'un élément.
2. Successeur, prédécesseur et supprimer supposent que l'on sait où se trouve l'élément.

Récapitulatif sur les files de priorités

Structure	max	extraire-max	augmenter [3]	insérer
Tableau non ordonné	$O(n)$	$O(n)$	$\Theta(1)$	$\Theta(1)^{\text{am}}$
Tableau ordonné	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$\Theta(1)$	$\Theta(1)$
Table de hachage	$O(n)$	$O(n)$	$\Theta(1)$	$\Theta(1)$
Tas (binaire)	$\Theta(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Tas de Fibonacci	$\Theta(1)$	$O(\log n)^{\text{am}}$	$\Theta(1)^{\text{am}}$	$\Theta(1)$
Arbre rech. non équilibré	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Arbre rech. équilibré	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

3. Augmenter suppose que l'on sait où se trouve l'élément.

Récapitulatif sur les algorithmes

Tas (34) Entasser-Max (10), Construire-Tas-Max (2), Tri-Par-Tas (5), Maximum-Tas (3), Extraire-Max-Tas (5), Augmenter-Clé-Tas (6), Insérer-Tas-Max (3).

Forêt d'ensembles disjoints (19) Créer-Ensemble (2), Union (9), Trouver-Ensemble (3), Composantes-Connexes (4), Même-Composante (1).

Arbre binaire de recherche (59) Parcours-Infixe (4), Arbre-Rechercher (6), Arbre-Minimum (3), Arbre-Maximum (3), Arbre-Successeur (7), Arbre-Insérer (15), Transplanter (8), Arbre-Supprimer (13).

Arbre rouge-noir (49) Rotation-Gauche (13), RN-Insérer (19), RN-Insérer-Correction (17).

B-arbre (64) Rechercher-B-Arbre (10), Créer-B-Arbre (5), Partager-Enfant-B-Arbre (20), Insérer-B-Arbre (11), Insérer-B-Arbre-Incomplet (18).

Arbre de rangs (14) Récupérer-Rang (7), Déterminer-Rang (7).

Arbre d'intervalles (7) Rechercher-Intervalle (7).

Récapitulatif sur les tris

Tri	pire cas	moyenne	meilleur cas	en place	stable	en ligne
insertion	$O(n^2)$	$O(n^2)$	$O(n)$	✓	✓	✓
bulles	$O(n^2)$	$O(n^2)$	$O(n)$	✓	✓	✗
fusion	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	✗	✓	✗
rapide	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	✓	✗	✗
tas	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	✓	✗	✗
arbre rech.	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	✗	✓	✓
arbre rech. équilibré	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	✗	✓	✓

On ne sait pas s'il existe un algorithme en $O(n \log n)$ en place et stable.

Plan

Résumé

Contenu

- La structure d'arbre peut s'étendre (ou s'augmenter) pour résoudre de nouveaux problèmes : ajout de champs supplémentaires ; adaptations des opérations ; nouvelles opérations.
- On peut déterminer les rangs dynamiquement.
- On peut faire des recherches sur les intervalles.

Prochaines échéances

- QCM à la prochaine séance de CM.
- Mini-projet : à rendre la semaine du 24/2.
- DS 1 : semaine du 24/3.

Résumé

Contenu

- La structure d'arbre peut s'étendre (ou s'augmenter) pour résoudre de nouveaux problèmes : ajout de champs supplémentaires ; adaptations des opérations ; nouvelles opérations.
- On peut déterminer les rangs dynamiquement.
- On peut faire des recherches sur les intervalles.

Prochaines échéances

- QCM à la prochaine séance de CM.
- Mini-projet : à rendre la semaine du 24/2.
- DS 1 : semaine du 24/3.