Grandchildren weight-balanced binary search trees

Vincent Jugé

Université Gustave Eiffel
CNRS & Université Paris-Cité

11/08/2025

Contents

Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?

Answer: Use a sorted array + dichotomy.

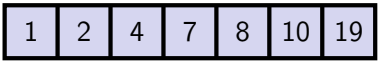
Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?

Answer: Use a sorted array + dichotomy.

$3 \in S$?



Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?

Answer: Use a sorted array + dichotomy.

$3 \in S$?

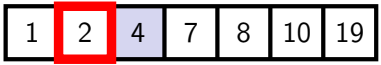
Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?

Answer: Use a sorted array + dichotomy.

$3 \in S$?



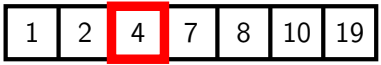
Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?

Answer: Use a sorted array + dichotomy.

$3 \in S$? **No!**



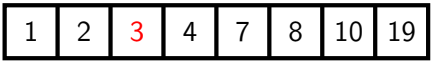
Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?
Query #2: Insert key k in my set S.

Answer: Use a sorted array + dichotomy.

Add 3!

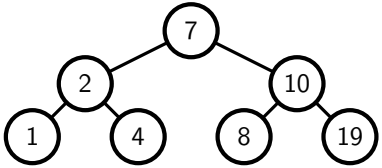


Maintaining ordered sets

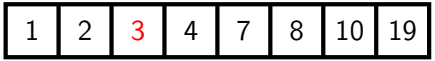
Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?
Query #2: Insert key k in my set S.

Answer: Use a sorted array + dichotomy. **Oops!**
Use a low-height binary search tree instead.



Add 3!

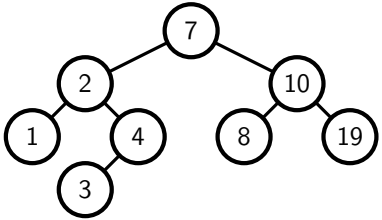


Maintaining ordered sets

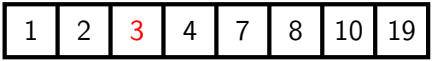
Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?
Query #2: Insert key k in my set S.

Answer: Use a sorted array + dichotomy. **Oops!**
Use a low-height binary search tree instead.



Add 3!

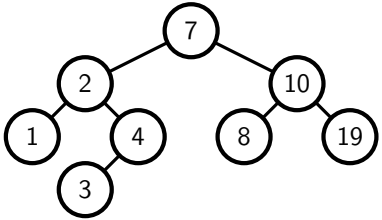


Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?
Query #2: Insert key k in my set S.
Query #3: Delete key k from my set S.

Answer: Use a sorted array + dichotomy. **Oops!**
Use a low-height binary search tree instead.



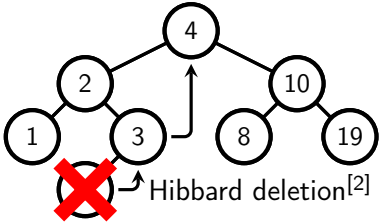
Remove 7!

Maintaining ordered sets

Which data structure should I use to manipulate ordered sets?

Query #1: Does key k belong to my set S?
Query #2: Insert key k in my set S.
Query #3: Delete key k from my set S.

Answer: Use a sorted array + dichotomy. **Oops!**
Use a low-height binary search tree instead.

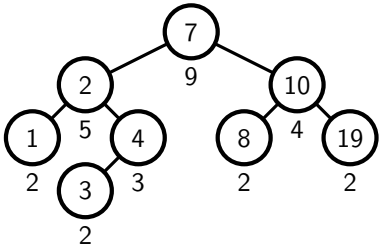


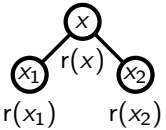
Remove 7!

Balanced binary search trees

Maintaining search trees of height $\mathcal{O}(\log(n))$ often requires some kind of **rank** and **balance**.

1. Weight-balanced trees[3]: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\qquad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\qquad (1/\sqrt{2} \leqslant \alpha < 9/11)$

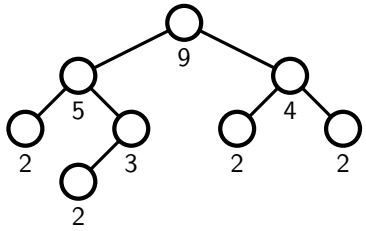


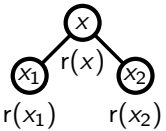


Balanced binary search trees

Maintaining search trees of height $\mathcal{O}(\log(n))$ often requires some kind of **rank** and **balance**.

1. Weight-balanced trees[3]: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\qquad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\qquad (1/\sqrt{2} \leqslant \alpha < 9/11)$

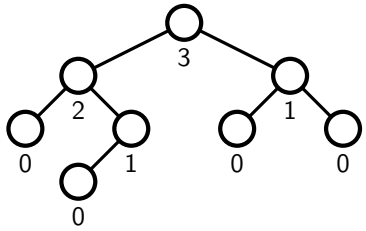


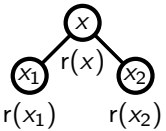


Balanced binary search trees

Maintaining search trees of height $\mathcal{O}(\log(n))$ often requires some kind of **rank** and **balance**.

1. Weight-balanced trees[3]: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\qquad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\qquad (1/\sqrt{2} \leqslant \alpha < 9/11)$
2. AVL trees[1]: $\mathsf{r}(x) = \mathsf{h}(x)$ $\qquad (\mathsf{r}(x) = \max\{\mathsf{r}(x_1), \mathsf{r}(x_2)\} + 1$ and $\mathsf{r}(\bot) = -1)$
   $\mathsf{r}(x) \leqslant \mathsf{r}(x_i) + 2$

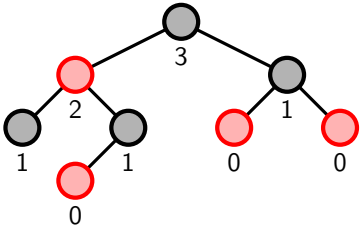


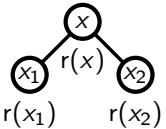


Balanced binary search trees

Maintaining search trees of height $\mathcal{O}(\log(n))$ often requires some kind of **rank** and **balance**.

1. Weight-balanced trees[3]: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\qquad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\qquad (1/\sqrt{2} \leqslant \alpha < 9/11)$
2. AVL trees[1]: $\mathsf{r}(x) = \mathsf{h}(x)$ $\qquad (\mathsf{r}(x) = \max\{\mathsf{r}(x_1), \mathsf{r}(x_2)\} + 1$ and $\mathsf{r}(\bot) = -1)$
   $\mathsf{r}(x) \leqslant \mathsf{r}(x_i) + 2$
3. Red-black trees[5,9]: $\mathsf{r}(x) = ?$ $\qquad (\mathsf{r}(\bot) = -1)$
   $\mathsf{r}(x_i) + 1 \leqslant \mathsf{r}(x) \leqslant \mathsf{r}(x_i) + 2$ $\qquad (x_i$ red $\Leftrightarrow \lfloor \mathsf{r}(x_i)/2 \rfloor = \lfloor \mathsf{r}(x)/2 \rfloor)$

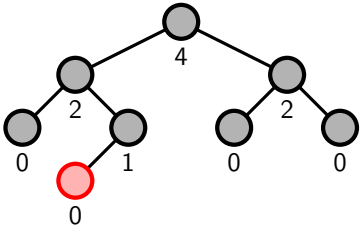


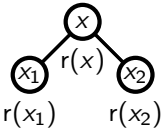


Balanced binary search trees

Maintaining search trees of height $\mathcal{O}(\log(n))$ often requires some kind of **rank** and **balance**.

1. Weight-balanced trees[3]: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\quad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\quad (1/\sqrt{2} \leqslant \alpha < 9/11)$
2. AVL trees[1]: $\mathsf{r}(x) = \mathsf{h}(x)$ $\quad (\mathsf{r}(x) = \max\{\mathsf{r}(x_1), \mathsf{r}(x_2)\} + 1$ and $\mathsf{r}(\bot) = -1)$
   $\mathsf{r}(x) \leqslant \mathsf{r}(x_i) + 2$
3. Red-black trees[5,9]: $\mathsf{r}(x) = ?$ $\quad (\mathsf{r}(\bot) = -1)$
   $\mathsf{r}(x_i) + 1 \leqslant \mathsf{r}(x) \leqslant \mathsf{r}(x_i) + 2$ $\quad (x_i$ red $\Leftrightarrow \lfloor \mathsf{r}(x_i)/2 \rfloor = \lfloor \mathsf{r}(x)/2 \rfloor)$

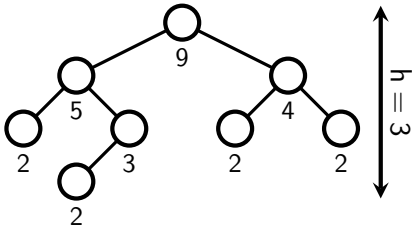


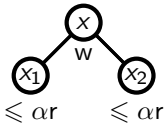


Height and internal/external path length

In weight-balanced trees with parameter $\alpha \in [1/\sqrt{2}, 9/11)$, in the worst case,

1. Height h $\approx \log_\alpha(n)$.



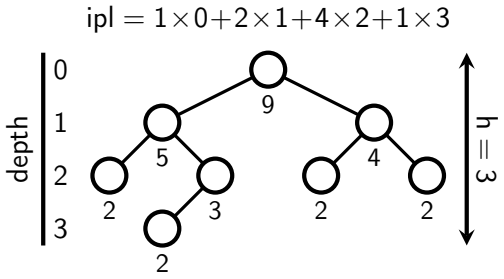


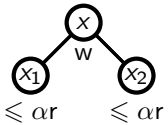
Height and internal/external path length

In weight-balanced trees with parameter $\alpha \in [1/\sqrt{2}, 9/11)$, in the worst case,

1. Height $h \approx \log_{\alpha}(n)$.
2. Internal path length[4] $ipl \approx n \log_2(n)/H_2(\alpha)$, where

$$H_2(\alpha) = -\alpha \log_2(\alpha) - (1-\alpha)\log_2(1-\alpha).$$



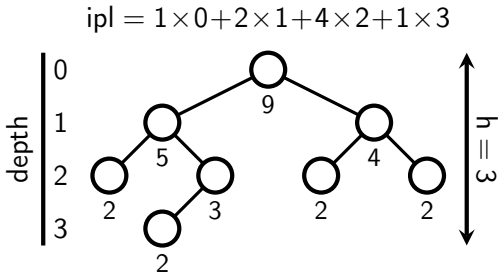


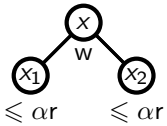
Height and internal/external path length

In weight-balanced trees with parameter $\alpha \in [1/\sqrt{2}, 9/11)$, in the worst case,

1. Height $h \approx \log_\alpha(n) = 2\log_2(n)$.
2. Internal path length[4] $ipl \approx n\log_2(n)/H_2(\alpha) \approx 1.146 n \log_2(n)$, where

$$H_2(\alpha) = -\alpha\log_2(\alpha) - (1-\alpha)\log_2(1-\alpha).$$

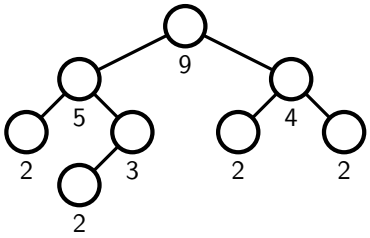


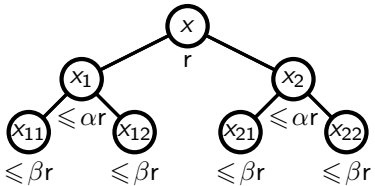


Contents

Grandchildren balanced trees

1. Grandchildren balanced trees: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\quad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\quad (1/\sqrt{2} \leqslant \alpha < 9/11)$
   $\mathsf{r}(x_{ij}) \leqslant \beta \mathsf{r}(x)$ $\quad (? \leqslant \beta \leqslant ?)$

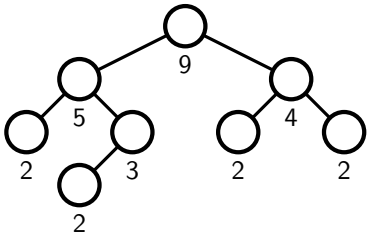


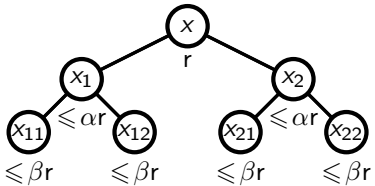


Grandchildren balanced trees

1. Grandchildren balanced trees: $\mathsf{r}(x) = \#\mathcal{T}(x) + 1$ $\quad (\mathsf{r}(x) = \mathsf{r}(x_1) + \mathsf{r}(x_2)$ and $\mathsf{r}(\bot) = 1)$
   $\mathsf{r}(x_i) \leqslant \alpha \mathsf{r}(x)$ $\quad (1/\sqrt{2} \leqslant \alpha \leqslant 3/4)$
   $\mathsf{r}(x_{ij}) \leqslant \beta \mathsf{r}(x)$ $\quad (\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2)$

$$\text{where } \mathscr{B}(\alpha) = \frac{\sqrt{1+4\alpha}-1}{2}.$$

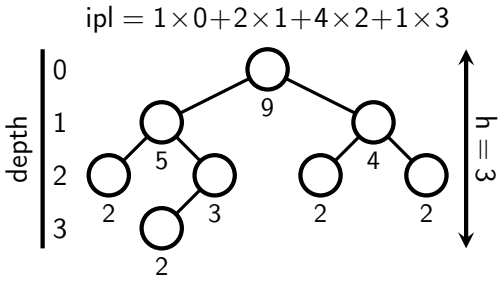


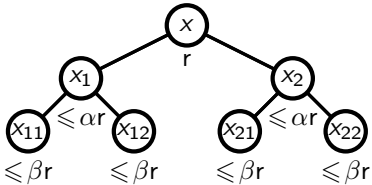


Improved height and internal/external path length

In trees with parameters $\alpha \in [1/\sqrt{2}, 3/4]$ and $\beta \in [\mathscr{B}(\alpha), \alpha^2]$, in the worst case,

1. Height $h \approx 2\log_\beta(n) \approx 1.880\log_2(n) < 2\log_2(n)$.



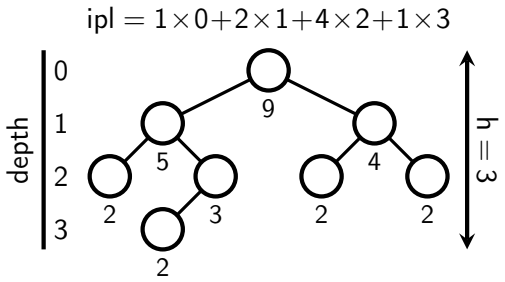


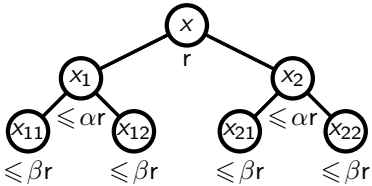
Improved height and internal/external path length

In trees with parameters $\alpha \in [1/\sqrt{2}, 3/4]$ and $\beta \in [\mathscr{B}(\alpha), \alpha^2]$, in the worst case,

1. Height $h \approx 2\log_\beta(n) \approx 1.880\log_2(n) < 2\log_2(n)$.
2. Internal path length $ipl \approx n\log_2(n)/\Delta \approx 1.127 n\log_2(n) < 1.146 n\log_2(n)$, where

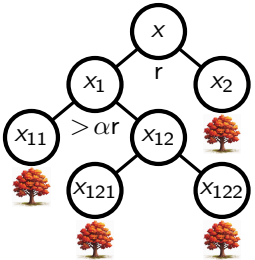
$$\Delta = \frac{H_2(\alpha) + \alpha H_2(\beta/\alpha)}{1+\alpha}.$$



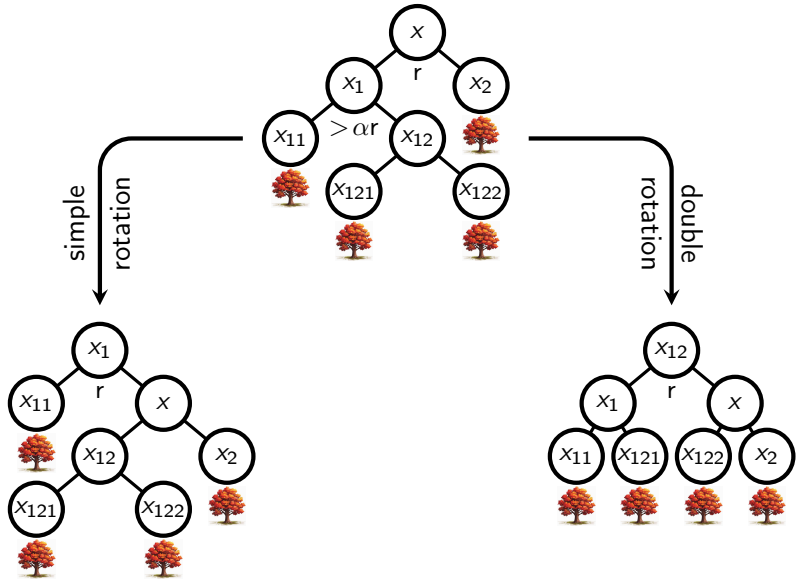


Contents

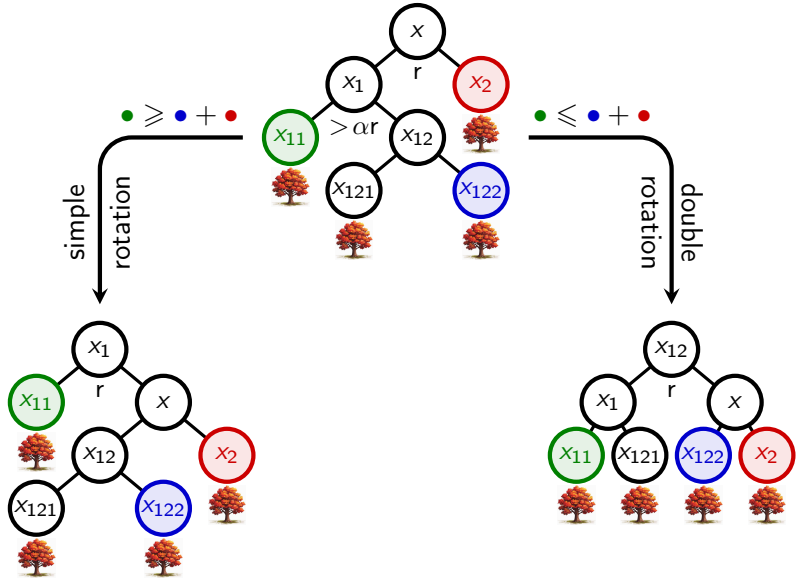
Routine **C**: Use rotations to keep your **C**hildren small[3]



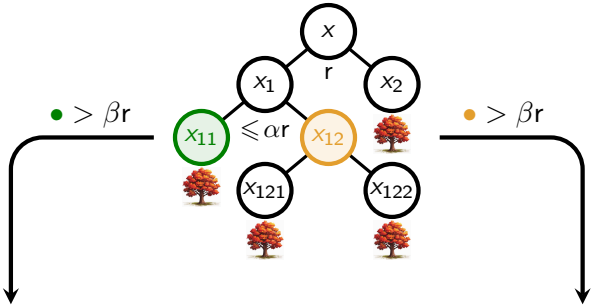
Routine **C**: Use rotations to keep your **C**hildren small[3]



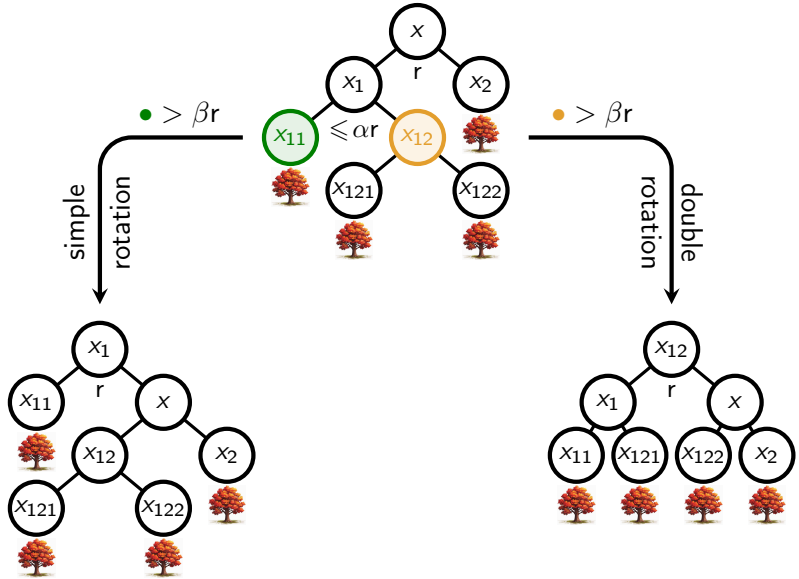
Routine **C**: Use rotations to keep your **C**hildren small[3]



Routine **GC**: Use rotations to keep your **G**rand**C**hildren small



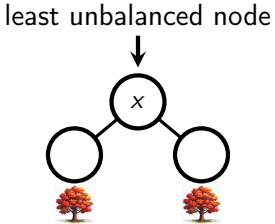
Routine **GC**: Use rotations to keep your **G**rand**C**hildren small



Combining both routines **C** and **GC**

Pros & cons of each routine:

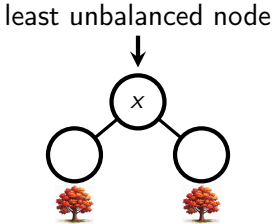
- Routine **C** ensures everybody has small children once your descendants have small children. It **may** mess with the grandchildren balance of those nodes its displaces. ☹



Combining both routines **C** and **GC**

Pros & cons of each routine:

- Routine **C** ensures everybody has small children once your descendants have small children. It **may** mess with the grandchildren balance of those nodes its displaces. ☹
- Routine **GC** ensures everybody has small grandchildren **once** everybody has small children. It does not mess with anybody. ☺



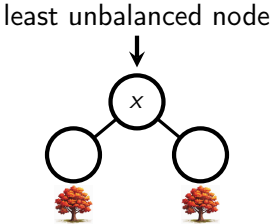
Combining both routines **C** and **GC**

Pros & cons of each routine:

- Routine **C** ensures everybody has small children once your descendants have small children. It **may** mess with the grandchildren balance of those nodes its displaces. ☹
- Routine **GC** ensures everybody has small grandchildren **once** everybody has small children. It does not mess with anybody. ☺

Recipe:

1. Launch routine **C** on x.



Combining both routines **C** and **GC**

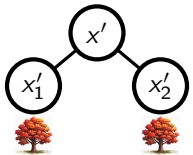
Pros & cons of each routine:

- Routine **C** ensures everybody has small children once your descendants have small children. It **may** mess with the grandchildren balance of those nodes its displaces. ☹
- Routine **GC** ensures everybody has small grandchildren **once** everybody has small children. It does not mess with anybody. ☺

Recipe:

1. Launch routine **C** on x.

   Our new root x' and its children x'_1 and x'_2 might have too large grandchildren.



Combining both routines **C** and **GC**

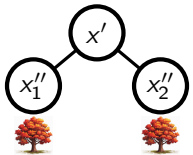
Pros & cons of each routine:

- Routine **C** ensures everybody has small children once your descendants have small children. It **may** mess with the grandchildren balance of those nodes its displaces. ☹
- Routine **GC** ensures everybody has small grandchildren **once** everybody has small children. It does not mess with anybody. ☺

Recipe:

1. Launch routine **C** on x.

   Our new root x' and its children x'_1 and x'_2 might have too large grandchildren.
2. Launch routine **GC** on x'_1 and x'_2 (in parallel).



Combining both routines **C** and **GC**

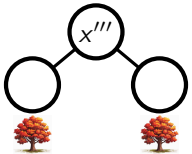
Pros & cons of each routine:

- Routine **C** ensures everybody has small children once your descendants have small children. It **may** mess with the grandchildren balance of those nodes its displaces. ☹
- Routine **GC** ensures everybody has small grandchildren **once** everybody has small children. It does not mess with anybody. ☺

Recipe:

1. Launch routine **C** on x.

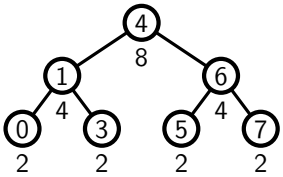
   Our new root x' and its children x'_1 and x'_2 might have too large grandchildren.
2. Launch routine **GC** on x'_1 and x'_2 (in parallel).
3. Launch routine **GC** on x'. ✓



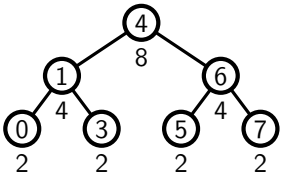
Contents

What is top-down maintenance?

Insert 2 **bottom-up**

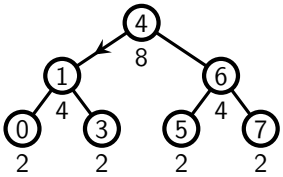


Insert 2 **top-down**

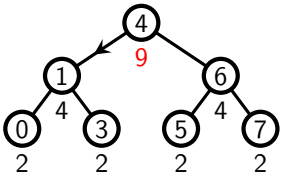


What is top-down maintenance?

Insert 2 **bottom-up**

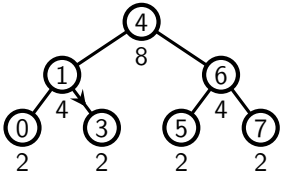


Insert 2 **top-down**

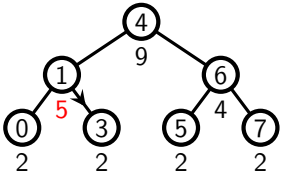


What is top-down maintenance?

Insert 2 **bottom-up**

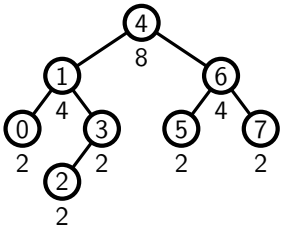


Insert 2 **top-down**

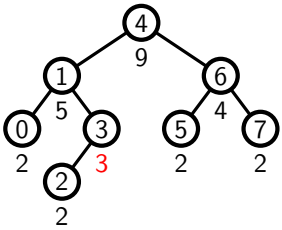


What is top-down maintenance?

Insert 2 **bottom-up**

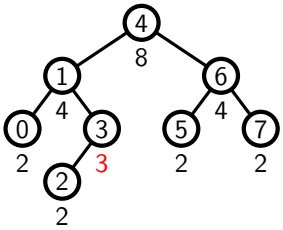


Insert 2 **top-down**

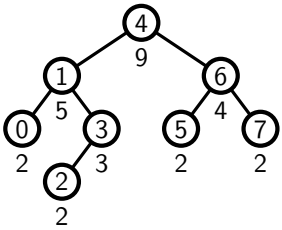


What is top-down maintenance?

Insert 2 **bottom-up**

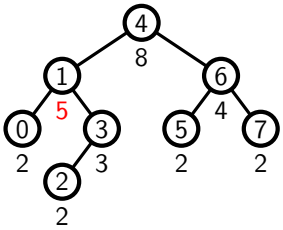


Insert 2 **top-down**

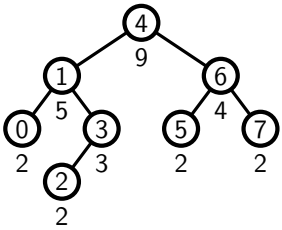


What is top-down maintenance?

Insert 2 **bottom-up**

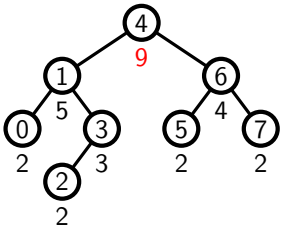


Insert 2 **top-down**

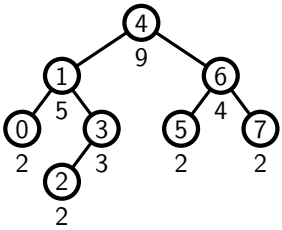


What is top-down maintenance?

Insert 2 **bottom-up**

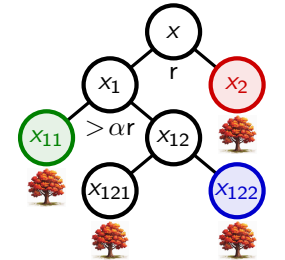


Insert 2 **top-down**

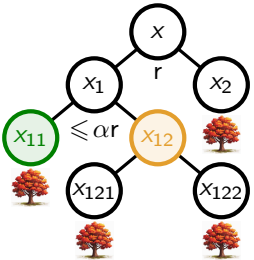


Prevent catastrophes just before they might occur!

Make **C** and **GC** routines trigger rotations **as soon as** adding or deleting a leaf below x might make x unbalanced[7].



$\bullet \leqslant \bullet + \bullet$ or $\bullet \geqslant \bullet + \bullet$



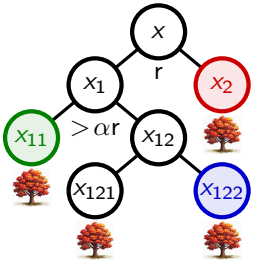
$\bullet > \beta\mathrm{r}$ or $\bullet > \beta\mathrm{r}$

Prevent catastrophes just before they might occur!

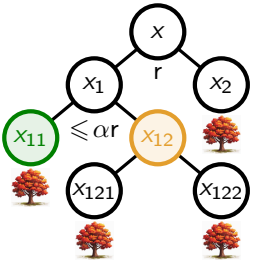
Make **C** and **GC** routines trigger rotations **as soon as** adding or deleting a leaf below x might make x unbalanced[7].

Redundant update = Trying to insert an already present key or delete an already absent key

At each node x, store $\mathrm{r}(x_i)$ for some child x_i **not** on the branch you modify[8]:
It just remains to restore the tree rank, instead of all node ranks on the modified branch.



$\bullet \leqslant \bullet + \bullet$ or $\bullet \geqslant \bullet + \bullet$



$\bullet > \beta\mathrm{r}$ or $\bullet > \beta\mathrm{r}$

Contents

Correctness and complexity bounds

Correctness:

Relies on **41** inequalities as lovely as $(1-\gamma)^2 \leqslant \gamma^2(1-\alpha)$ whenever $\frac{1}{\sqrt{2}} \leqslant \alpha \leqslant \frac{3}{4}$ and

$$\gamma = \frac{2\alpha}{1+\alpha+\sqrt{1+\alpha^2+4\alpha^3-2\alpha\sqrt{1+4\alpha}}}.$$

Correctness and complexity bounds

Correctness:

Relies on **41** inequalities as lovely as $(1-\gamma)^2 \leqslant \gamma^2(1-\alpha)$ whenever $\frac{1}{\sqrt{2}} \leqslant \alpha \leqslant \frac{3}{4}$ and

$$\gamma = \frac{2\alpha}{1+\alpha+\sqrt{1+\alpha^2+4\alpha^3-2\alpha\sqrt{1+4\alpha}}}.$$

Complexity[6]:

Answering q queries requires $\mathcal{O}(q/\mathsf{c})$ rotations, where $\mathsf{c} = \min\{\alpha - 1/\sqrt{2}, \beta - \mathscr{B}(\alpha), \alpha^2 - \beta\}$.

Correctness and complexity bounds

Correctness:

Relies on **41** inequalities as lovely as $(1-\gamma)^2 \leqslant \gamma^2(1-\alpha)$ whenever $\frac{1}{\sqrt{2}} \leqslant \alpha \leqslant \frac{3}{4}$ and

$$\gamma = \frac{2\alpha}{1+\alpha+\sqrt{1+\alpha^2+4\alpha^3-2\alpha\sqrt{1+4\alpha}}}.$$

Complexity[6]:

Answering q queries requires $\mathcal{O}(q/\mathsf{c})$ rotations, where $\mathsf{c} = \min\{\alpha - 1/\sqrt{2}, \beta - \mathscr{B}(\alpha), \alpha^2 - \beta\}$.

Proof idea:

Give each node x a counter whose value

- increases by $1/\mathsf{r}(x)$ when a leaf below x is inserted/deleted,
- is reset to 0 when a rotation changes the vicinity of x.

Correctness and complexity bounds

Correctness:

Relies on **41** inequalities as lovely as $(1-\gamma)^2 \leqslant \gamma^2(1-\alpha)$ whenever $\frac{1}{\sqrt{2}} \leqslant \alpha \leqslant \frac{3}{4}$ and

$$\gamma = \frac{2\alpha}{1+\alpha+\sqrt{1+\alpha^2+4\alpha^3-2\alpha\sqrt{1+4\alpha}}}.$$

Complexity[6]:

Answering q queries requires $\mathcal{O}(q/\mathsf{c})$ rotations, where $\mathsf{c} = \min\{\alpha - 1/\sqrt{2}, \beta - \mathscr{B}(\alpha), \alpha^2 - \beta\}$.

Proof idea:

Give each node x a counter whose value

- increases by $1/\mathsf{r}(x)$ when a leaf below x is inserted/deleted,
- is reset to 0 when a rotation changes the vicinity of x.

The sum of all counters

- increases by $\mathcal{O}(1)$ when a leaf is inserted/deleted;
- decreases by $\Omega(\mathsf{c})$ when a rotation takes place.

Bibliography

1. G. Adel'son-Velskii & E. Landis, *An algorithm for organization of information* 1962
2. T. Hibbard, *Some combinatorial properties of certain trees with applications to searching and sorting* 1962
3. J. Nievergelt & E. Reingold, *Binary search trees of bounded balance* 1972
4. J. Nievergelt & C. K. Wong, *Upper bounds for the total path length of binary trees* 1973
5. L. Guibas & R. Sedgewick, *A dichromatic framework for balanced trees* 1978
6. N. Blum & K. Mehlhorn, *Average number of rebalancing operations in weight-balanced trees* 1980
7. T. Lai & D. Wood, *A top-down updating algorithm for weight-balanced trees* 1993
8. C. Martínez & S. Roura, *Randomized binary search trees* 1998
9. C. B. Haeupler, S. Sen & R. Tarjan, *Rank-balanced trees* 2015


MERCI POUR VOTRE ATTENTION !
NE POSEZ PAS DE QUESTIONS
TROP DIFFICILES S'IL VOUS PLAÎT !