11 Augmentierung von Datenstrukturen

Ziel: entwickle Datenstruktur mit Operationen:

- **Insert(x)**: füge x ein.
- **Search(k)**: suche nach Element mit Schlüssel k.
- **Delete(x)**: Lösche Element, das durch pointer/handle x referenziert wird.
- **find-by-rank(ℓ)**: gibt das ℓ-kleinste Element zurück. `NULL` falls die Datenstruktur weniger als ℓ Elemente enthält.

Abwandlung/Augmentierung einer existierenden Datenstruktur anstatt eine neue zu entwickeln.

11 Augmentierung von Datenstrukturen

Ziel: entwickle Datenstruktur mit Operationen:

- **Insert(x)**: füge x ein.
- **Search(k)**: suche nach Element mit Schlüssel k.
- **Delete(x)**: Lösche Element, das durch pointer/handle x referenziert wird.
- **find-by-rank(ℓ)**: gibt das ℓ-kleinste Element zurück. NULL falls die Datenstruktur weniger als ℓ Elemente enthält.

Abwandlung/Augmentierung einer existierenden Datenstruktur anstatt eine neue zu entwickeln.

11 Augmentierung von Datenstrukturen

Wie augmentiert man eine Datenstruktur?

1. wähle eine zugrundeliegende Datenstruktur.
2. entscheide welche zusätzliche Information die Struktur speichern soll.
3. zeige, dass man diese zusätzlichen Informationen effizient updaten kann wenn sich die Datenstruktur ändert (z.B. beim Einfügen/Löschen)
4. entwickle die zusätzlichen Operationen

- Natürlich hängen diese Schritte voneinander ab. Es macht z.B. keinen Sinn Informationen zu speichern die man entweder nicht effizient updaten kann, oder nicht genügend Informationen um die neuen Operationen zu unterstützen.
- Allerdings kann man die obige Reihenfolge benutzen um die neue Datenstruktur zu dokumentieren.

11 Augmentierung von Datenstrukturen

Wie augmentiert man eine Datenstruktur?

1. wähle eine zugrundeliegende Datenstruktur.
2. entscheide welche zusätzliche Information die Struktur speichern soll.
3. zeige, dass man diese zusätzlichen Informationen effizient updaten kann wenn sich die Datenstruktur ändert (z.B. beim Einfügen/Löschen)
4. entwickle die zusätzlichen Operationen

- Natürlich hängen diese Schritte voneinander ab. Es macht z.B. keinen Sinn Informationen zu speichern die man entweder nicht effizient updaten kann, oder nicht genügend Informationen um die neuen Operationen zu unterstützen.
- Allerdings kann man die obige Reihenfolge benutzen um die neue Datenstruktur zu dokumentieren.

11 Augmentierung von Datenstrukturen

Wie augmentiert man eine Datenstruktur?

1. wähle eine zugrundeliegende Datenstruktur.
2. entscheide welche zusätzliche Information die Struktur speichern soll.
3. zeige, dass man diese zusätzlichen Informationen effizient updaten kann wenn sich die Datenstruktur ändert (z.B. beim Einfügen/Löschen)
4. entwickle die zusätzlichen Operationen

- Natürlich hängen diese Schritte voneinander ab. Es macht z.B. keinen Sinn Informationen zu speichern die man entweder nicht effizient updaten kann, oder nicht genügend Informationen um die neuen Operationen zu unterstützen.
- Allerdings kann man die obige Reihenfolge benutzen um die neue Datenstruktur zu dokumentieren.

11 Augmentierung von Datenstrukturen

Wie augmentiert man eine Datenstruktur?

1. wähle eine zugrundeliegende Datenstruktur.
2. entscheide welche zusätzliche Information die Struktur speichern soll.
3. zeige, dass man diese zusätzlichen Informationen effizient updaten kann wenn sich die Datenstruktur ändert (z.B. beim Einfügen/Löschen)
4. entwickle die zusätzlichen Operationen

- Natürlich hängen diese Schritte voneinander ab. Es macht z.B. keinen Sinn Informationen zu speichern die man entweder nicht effizient updaten kann, oder nicht genügend Informationen um die neuen Operationen zu unterstützen.
- Allerdings kann man die obige Reihenfolge benutzen um die neue Datenstruktur zu dokumentieren.

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

1. Wir wählen einen AVL-Baum als zugrundeliegende Datenstruktur.
2. Wir speichern in jedem Knoten v die Größe des zugehörigen Teilbaums.
3. Wir müssen in der Lage sein, die Größenfelder in jedem Knoten zu aktualisieren ohne die asymptotische Laufzeit von insert, delete, und search zu beeinträchtigen. Später...

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

1. Wir wählen einen AVL-Baum als zugrundeliegende Datenstruktur.
2. Wir speichern in jedem Knoten v die Größe des zugehörigen Teilbaums.
3. Wir müssen in der Lage sein, die Größenfelder in jedem Knoten zu aktualisieren ohne die asymptotische Laufzeit von insert, delete, und search zu beeinträchtigen. Später...

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

1. Wir wählen einen AVL-Baum als zugrundeliegende Datenstruktur.
2. Wir speichern in jedem Knoten v die Größe des zugehörigen Teilbaums.
3. Wir müssen in der Lage sein, die Größenfelder in jedem Knoten zu aktualisieren ohne die asymptotische Laufzeit von insert, delete, und search zu beeinträchtigen. Später...

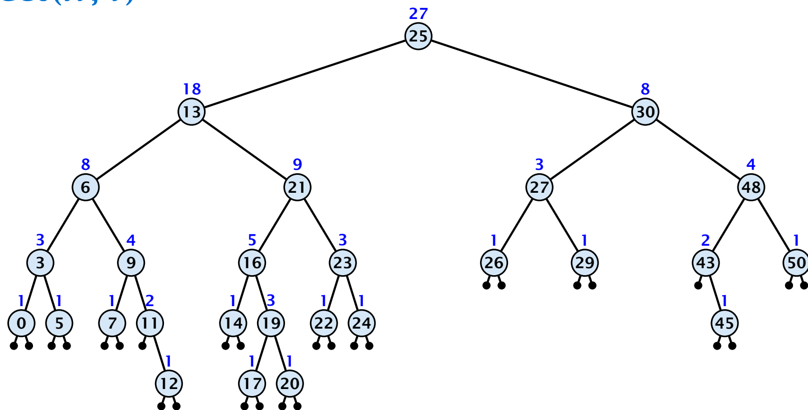
11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

4. Wie funktioniert find-by-rank?
   Find-by-rank(k) := Select(root, k) mit

```
1  Input: Zeiger auf Wurzel eines Teilbaums
2         gib i-kleinstes Element des Teilbaums aus
3
4  Select(x,i)
5      if (x == NULL) return NULL;
6      if (x->left != NULL) r = x->left->size+1; else r = 1;
7      if (i == r) return x;
8      if (i < r)
9          return Select(x->left,i);
10     else
11         return Select(x->right,i-r);
```

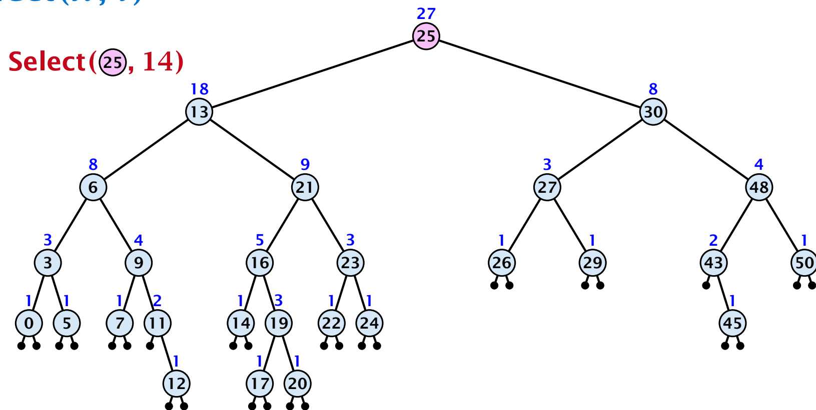
Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

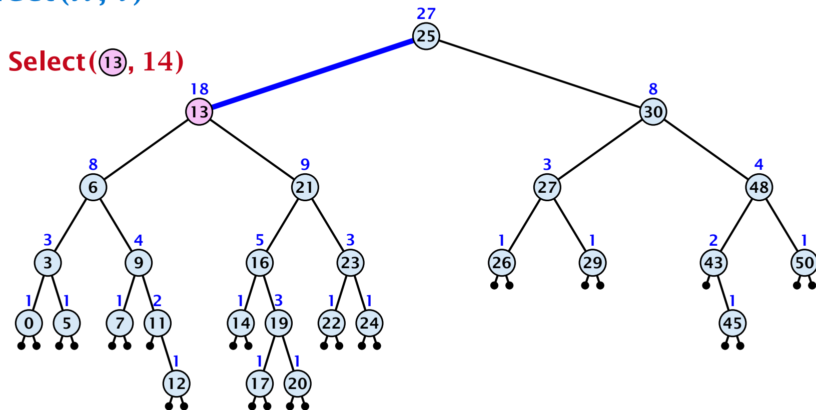
Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

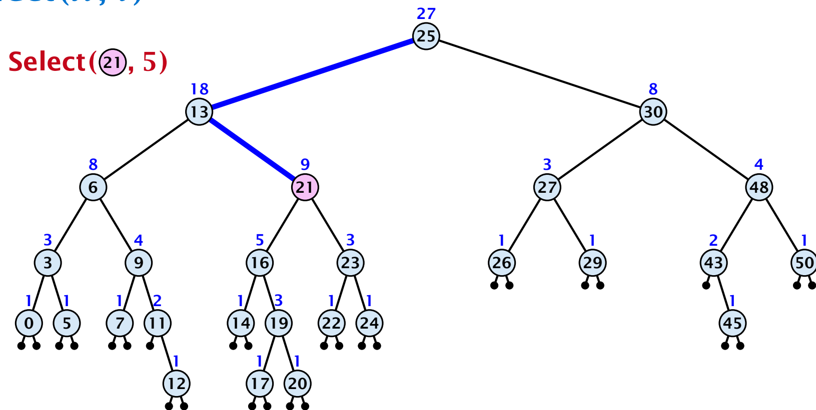
Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

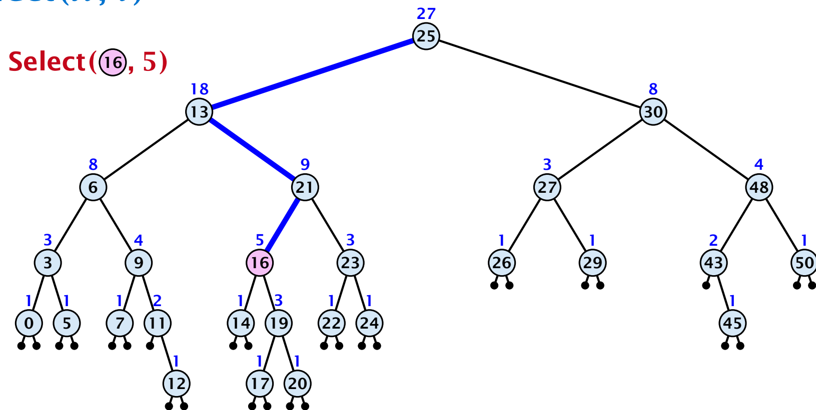
Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

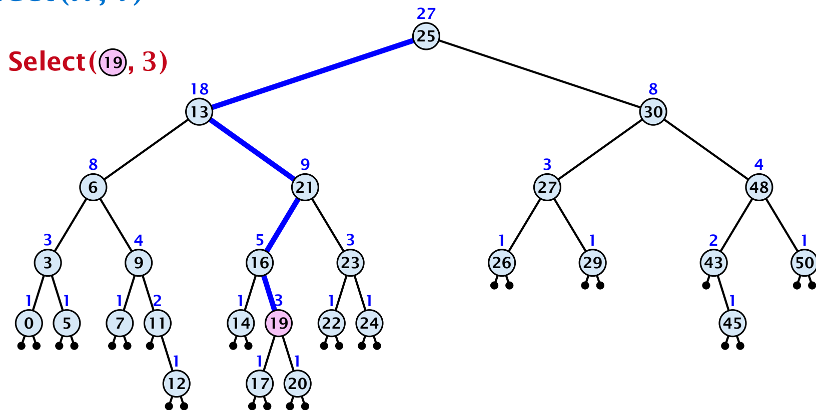
Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

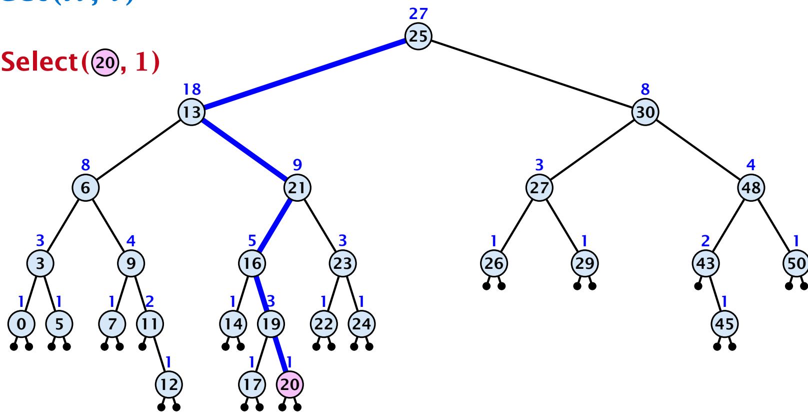
Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

Select(x, i)



Find-by-rank:

- entscheide ob man in dem linken oder rechten Teilbaum weitersuchen muss
- passe den index des Elementes an nach dem gesucht werden muss falls man rechts verzweigt.

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

3. Wie wird die Information aktualisiert?

Search(k): Nichts zu tun.

Insert(x): Während man nach der Einfügeposition sucht erhöht man das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

Delete(x): Wenn man ein Knoten überbrückt läuft man vom überbrückten Knoten aufwärts im Baum und reduziert das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

3. Wie wird die Information aktualisiert?

Search(k): Nichts zu tun.

Insert(x): Während man nach der Einfügeposition sucht erhöht man das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

Delete(x): Wenn man ein Knoten überbrückt läuft man vom überbrückten Knoten aufwärts im Baum und reduziert das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

3. Wie wird die Information aktualisiert?

Search(k): Nichts zu tun.

Insert(x): Während man nach der Einfügeposition sucht erhöht man das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

Delete(x): Wenn man ein Knoten überbrückt läuft man vom überbrückten Knoten aufwärts im Baum und reduziert das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

11 Augmentierung von Datenstrukturen

Ziel: Datenstruktur mit insert, delete, search, und find-by-rank mit Laufzeit $\mathcal{O}(\log n)$.

3. Wie wird die Information aktualisiert?

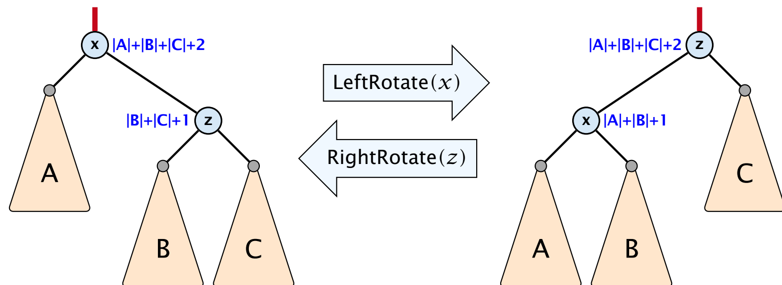
Search(k): Nichts zu tun.

Insert(x): Während man nach der Einfügeposition sucht erhöht man das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

Delete(x): Wenn man ein Knoten überbrückt läuft man vom überbrückten Knoten aufwärts im Baum und reduziert das Größenfeld jedes besuchten Knotens um 1. Größenfeld bei Rotationen aktualisieren!

Rotationen

Die einzige Operationen während der fix-up Routinen, die den Baum verändert und eine Aktualisierung der Größenfelder benötigt.



Die Knoten x und z sind die einzigen Knoten die ihre Größenfelder ändern.

Die neuen Größenfelder können lokal aus den Größenfelder der Kinder berechnet werden.