3 Vorrangwarteschlangen (priority queues)

3.1 Einleitung

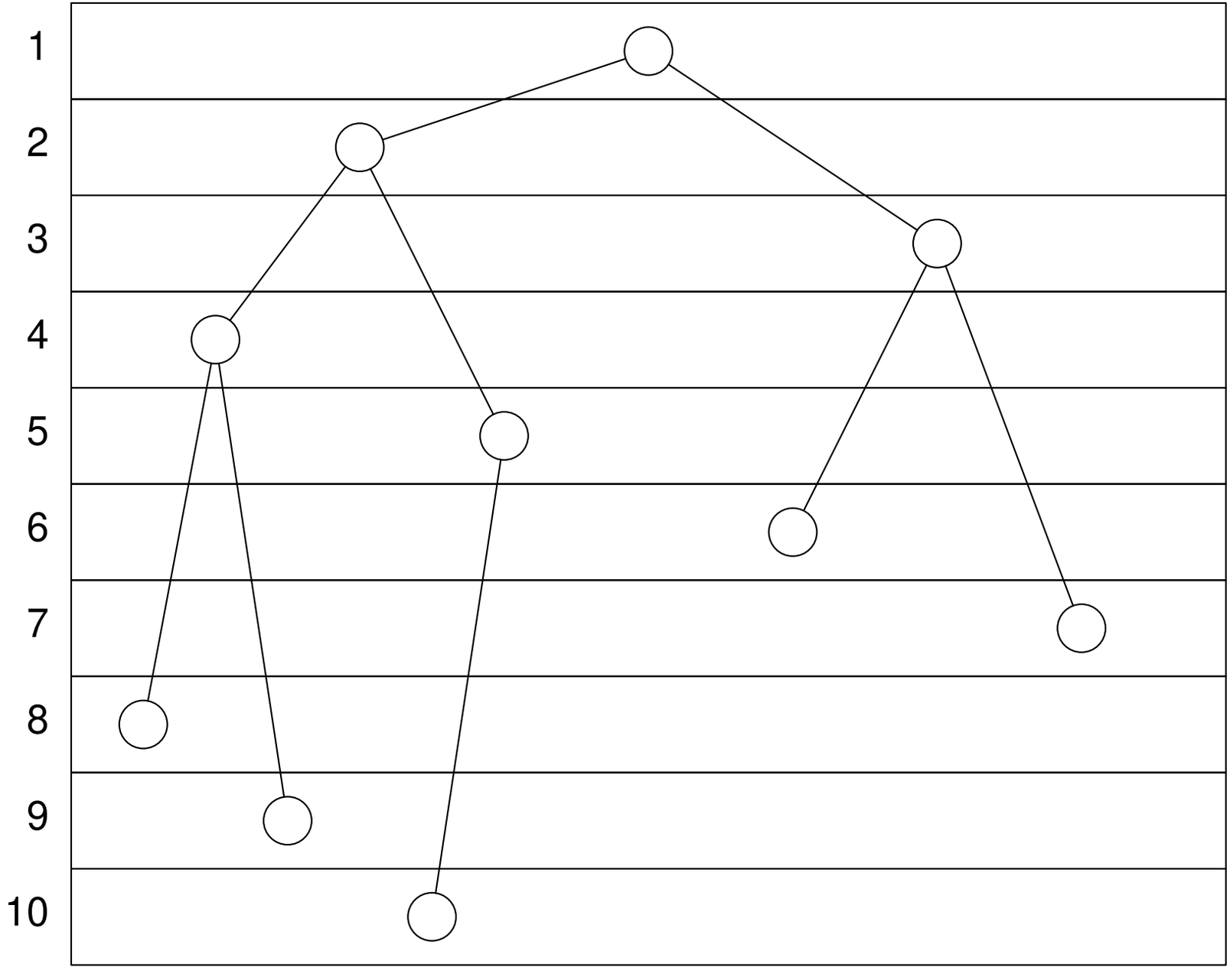
- Eine *Maximum-Vorrangwarteschlange* ist eine Datenstruktur, die folgende Operationen möglichst effizient unterstützt:
  - Einfügen eines Objekts mit einer bestimmten Priorität
  - Auslesen und ggf. Entnehmen eines Objekts mit maximaler Priorität
  - Nachträgliches Ändern der Priorität eines Objekts
  - Entfernen eines Objekts
  - Eventuell zusätzlich: Vereinigen zweier Warteschlangen
- Eine *Minimum-Vorrangwarteschlange* unterstützt entsprechend das Auslesen und ggf. Entnehmen eines Objekts mit minimaler statt maximaler Priorität.
- Anwendungsbeispiele:
  - Prozess-Disponent (scheduler) eines Betriebssystems
  - Huffman-Kodierung (vgl. § 4.3)
  - Bestimmte Graphalgorithmen (vgl. § 5.5.8 und § 5.6.5)

3.2 Binäre Halden (binary heaps)

3.2.1 Interpretation einer Reihe als Binärbaum

- Element $1 = 1_2$: Wurzelknoten
- Element $2 = 10_2 = 2 \cdot 1 + 0$: Linker Nachfolger des Wurzelknotens
- Element $3 = 11_2 = 2 \cdot 1 + 1$: Rechter Nachfolger des Wurzelknotens
- Element $4 = 100_2 = 2 \cdot 2 + 0$:
  Linker Nachfolger des linken Nachfolgers des Wurzelknotens
- Element $5 = 101_2 = 2 \cdot 2 + 1$:
  Rechter Nachfolger des linken Nachfolgers des Wurzelknotens
- Element $6 = 110_2 = 2 \cdot 3 + 0$:
  Linker Nachfolger des rechten Nachfolgers des Wurzelknotens
- Element $7 = 111_2 = 2 \cdot 3 + 1$:
  Rechter Nachfolger des rechten Nachfolgers des Wurzelknotens
- Usw.

Beispiel



Allgemein

- Zu einem Element i ist
  - Element $2i + 0$ sein linker Nachfolger (falls $2i + 0 \leq m$)
  - Element $2i + 1$ sein rechter Nachfolger (falls $2i + 1 \leq m$)
  - Element $\left\lfloor \frac{i}{2} \right\rfloor$ sein Vorgänger (falls $i > 1$)
- Dabei bezeichnet m die Gesamtzahl der Elemente des Baums, die auch kleiner als die Größe N der Reihe sein kann.
- Ebene l ($l = 0, 1, \ldots$) enthält die Elemente 2^l einschließlich bis 2^{l+1} ausschließlich (jedoch bis maximal m einschließlich).
- Damit befindet sich Element i auf Ebene $\lfloor \log_2 i \rfloor$.

Anmerkungen

- Die Multiplikation $2i$ ist effizient als Verschiebung des Bitmusters von i um eine Position nach links (`i << 1` in C, C++, Java, ...) berechenbar.
- Die ganzzahlige Division $\left\lfloor \frac{i}{2} \right\rfloor$ ist effizient als Verschiebung des Bitmusters von i um eine Position nach rechts (`i >> 1` in C, C++, Java, ...) berechenbar.

3.2.2 Minimum- und Maximum-Halden

- Definition:
  Ein Baum erfüllt die *Maximum-Bedingung*, wenn jeder seiner Knoten außer der Wurzel höchstens so groß wie sein Vorgänger ist.

- Folgerung:
  Alle Knoten eines Teilbaums sind höchstens so groß wie die Wurzel dieses Teilbaums.

- Insbesondere:
  Die Wurzel des gesamten Baums ist ein maximales Element des Baums.

- Definition:
  Die ersten *m* Elemente einer Reihe stellen eine *binäre Maximum-Halde* dar, wenn der durch diese Elemente definierte Binärbaum die Maximum-Bedingung erfüllt.

- *Minimum-Bedingung* und *Minimum-Halde* sind analog definiert.

3.2.3 Operationen auf Maximum-Halden (auf Minimum-Halden analog)

Absenken eines Elements

- Vorbedingung:
  Linker und rechter Teilbaum (falls vorhanden) des Elements i erfüllen die Maximum-Bedingung.

- Nachbedingung/Ziel:
  Der Teilbaum mit Wurzelknoten i erfüllt die Maximum-Bedingung.

- Ablauf:

  1 Wenn Knoten i keine Nachfolger besitzt, ist nichts zu tun.

  2 Wenn er genau einen (d. h. einen linken) Nachfolger besitzt, ist dies natürlich der größte Nachfolger.

  3 Wenn er zwei Nachfolger besitzt, bestimme den größeren von ihnen.
  (Falls beide gleich sind, wähle willkürlich einen von beiden.)

  4 Wenn Element i kleiner als dieser größte Nachfolger ist, vertausche es mit ihm und wiederhole dann die gesamte Operation für diesen Nachfolger.

- Die Laufzeit der Operation ist offenbar proportional zur Tiefe t des betrachteten Teilbaums und damit höchstens $O(\log_2 m)$.

Herstellen einer Maximum-Halde

- Gegeben: Eine Reihe der Größe N mit beliebigen Elementen.

- Nachbedingung/Ziel:
  Die ersten m Elemente der Reihe stellen eine Maximum-Halde dar.

- Ablauf:

  Für $i = \left\lfloor \frac{m}{2} \right\rfloor, \ldots, 1$:

  Führe die zuvor beschriebene Operation „Absenken“ für das Element i aus.

- Laufzeit
  - Die Tiefe des gesamten Baums ist $t = \lfloor \log_2 m \rfloor$.
  - Ein Teilbaum, dessen Wurzelknoten sich auf Ebene $l = 0, 1, \ldots$ befindet, hat höchstens Tiefe $t - l$. Dementsprechend ist die Laufzeit der Operation „Absenken“ für einen solchen Teilbaum $O(t - l)$.
  - Auf Ebene $l = 0, \ldots, t - 1$ gibt es höchstens 2^l abzusenkende Knoten. (Auf der untersten Ebene t werden keine Knoten abgesenkt.)
  - Also ist die Gesamtlaufzeit höchstens:

$$\sum_{l=0}^{t-1} 2^l \cdot (t-l) \underset{k=t-l}{=} \sum_{k=1}^{t} 2^{t-k} \cdot k = 2^t \cdot \sum_{k=1}^{t} k \cdot \left(\frac{1}{2}\right)^k \leq m \cdot \sum_{k=1}^{\infty} k \cdot \left(\frac{1}{2}\right)^k = m \cdot \frac{\frac{1}{2}}{\left(1-\frac{1}{2}\right)^2} = 2m$$

(Beachte: Für $|q| < 1$ gilt: $\sum_{k=1}^{\infty} k \cdot q^k = \frac{q}{(1-q)^2}$.)

Sortieren einer Reihe

- Gegeben: Eine Reihe der Größe N mit beliebigen Elementen.
- Nachbedingung/Ziel: Die Elemente der Reihe sind aufsteigend sortiert.
- Ablauf:
  1. Setze $m = N$
  2. Führe die Operation „Herstellen einer Maximum-Halde“ aus.
  3. Solange $m > 1$ ist:
     1. Vertausche Element 1 und m der Reihe.
     2. Verkleinere die Halde um ein Element, d. h. erniedrige m um 1.
     3. Führe die Operation „Absenken“ für Element 1 aus.
- Laufzeit
  - Es findet ein Aufruf der Operation „Herstellen einer Maximum-Halde“ mit Laufzeit $O(N)$ sowie $N - 1 = O(N)$ Aufrufe der Operation „Absenken“ statt, deren Laufzeit jeweils höchstens $O(\log N)$ beträgt.
  - Damit ergibt sich eine Gesamtlaufzeit von $O(N) + O(N) \cdot O(\log N) = O(N \log N)$.

3.2.4 Implementierung von Vorrangwarteschlangen

- Eine Maximum-Vorrangwarteschlange kann wie folgt durch eine binäre Maximum-Halde implementiert werden (und analog kann eine Minimum-Vorrangwarteschlange durch eine Minimum-Halde implementiert werden).

- N bezeichne die maximale Kapazität der Warteschlange, d. h. die Größe der zugrundeliegenden Reihe.

- m bezeichne die momentane Länge der Warteschlange, d. h. die Anzahl der momentan gespeicherten Objekte.
  Dies ist die gleichzeitig die momentane Größe der Halde.

Hilfsoperationen

- Anheben eines Elements

  Wenn das Element einen Vorgänger besitzt
  und seine Priorität größer als die des Vorgängers ist:

  1. Vertausche das Element mit seinem Vorgänger.
  2. Wiederhole die Operation für den Vorgänger.

- Verschieben eines Elements

  1. Führe die Operation „Anheben“ für das Element aus.
  2. Führe die Operation „Absenken“ für das Element aus (wobei sich die dabei durchgeführten Vergleiche auf die Prioritäten der Elemente beziehen).

  (Beachte, dass jede der beiden Operationen auch wirkungslos sein kann.)

Operationen

- Einfügen eines Objekts mit einer bestimmten Priorität
  1. Wenn m gleich N ist: Fehler, weil die Halde bereits voll ist.
  2. Andernfalls vergrößere die Halde um ein Element, d. h. erhöhe m um 1, und speichere das neue Objekt an Position m.
  3. Führe die Hilfsoperation „Anheben“ für das Element an Position m aus.
- Auslesen eines Objekts mit maximaler Priorität
  1. Wenn m gleich 0 ist: Fehler, weil die Halde leer ist.
  2. Andernfalls liefere das Objekt an Position 1 der Halde.

- Entnehmen eines Objekts mit maximaler Priorität
  1. Wenn m gleich 0 ist: Fehler, weil die Halde leer ist.
  2. Andernfalls merke das Objekt an Position 1 der Halde, um es später als Resultat liefern zu können.
  3. Versetze das Objekt an Position m der Halde an Position 1 und verkleinere die Halde um ein Element, d. h. erniedrige m um 1.
  4. Führe die Operation „Absenken“ für das Element an Position 1 aus (die wirkungslos ist, falls dieses Objekt durch die Verkleinerung nicht mehr zur Halde gehört).
  5. Liefere das anfangs gemerkte Objekt zurück.
- Nachträgliches Ändern der Priorität eines Objekts
  1. Ändere die Priorität des Objekts.
  2. Führe die Operation „Verschieben“ für das Objekt aus.

- Entfernen eines Objekts
  1. Ersetze das zu entfernende Objekt durch das Objekt an Position m der Halde und verkleinere die Halde um ein Element, d. h. erniedrige m um 1.
  2. Führe die Operation „Verschieben“ für das ersetzte Element aus (die wirkungslos ist, falls dieses Objekt durch die Verkleinerung nicht mehr zur Halde gehört).

Laufzeit der Operationen

- Das Auslesen eines Objekts mit maximaler Priorität benötigt offensichtlich Laufzeit $O(1)$.

- Alle anderen Operationen verschieben ein Objekt innerhalb der Halde ggf. um eine oder mehrere Ebenen nach oben oder unten.
  Dementsprechend ist ihre Laufzeit höchstens so groß wie die Tiefe des Baums, d. h. $O(\log m)$.

3.2.5 Praktisches Problem

- Wie finden die Operationen „Ändern der Priorität“ und „Entfernen“ innerhalb der Halde effizient (d. h. ohne sequentielle Suche) das jeweilige Objekt?
- Erste Idee:
  - Die Operation „Einfügen“ liefert die Position in der Halde zurück, an der sie das Objekt gespeichert hat.
  - Die Operationen „Ändern der Priorität“ und „Entfernen“ erhalten diese Position anstelle des Objekts selbst.
- Neues Problem: Die Position eines Objekts ist nicht stabil, weil viele Operationen Objekte in der Halde vertauschen.
- Lösung (wie für viele andere Informatik-Probleme) durch eine zusätzliche Indirektion:
  - In der Halde befinden sich nur Zeiger auf Objekte, die neben ihrer Priorität und irgendwelchen zusätzlichen Daten ihre aktuelle Position in der Halde enthalten.
  - Die Operation „Einfügen“ liefert einen Zeiger auf das eingefügte Objekt zurück.
  - Die Operationen „Ändern der Priorität“ und „Entfernen“ erhalten solche Zeiger als Parameter und kennen damit indirekt auch die Position des jeweiligen Objekts in der Halde.
  - Wenn (Zeiger auf) Objekte in der Halde vertauscht werden, werden die in ihnen gespeicherten Positionen jeweils aktualisiert.

3.3 Binomial-Halden (binomial heaps)

3.3.1 Binomial-Bäume

Definition

- Ein *Binomial-Baum* mit Grad 0 besteht aus einem einzelnen Knoten.
- Für $k \geq 1$ entsteht ein *Binomial-Baum* mit Grad k aus zwei Binomial-Bäumen mit Grad $k-1$, von denen einer zu einem Nachfolger (üblicherweise zum ersten Nachfolger) des Wurzelknotens des anderen gemacht wird.

Eigenschaften von Binomial-Bäumen

Für jeden Binomial-Baum mit Grad $k \in \mathbb{N}_0$ gilt:

1. Die Tiefe des Baums ist k.
2. Der Grad seines Wurzelknotens ist k.
3. Der Grad aller anderen Knoten ist kleiner als k.
4. Die Nachfolger des Wurzelknotens sind Binomial-Bäume mit Grad $k-1, \ldots, 0$ (üblicherweise in dieser Reihenfolge).
5. Der Baum besitzt 2^k Knoten.
6. Auf Ebene l ($l = 0, \ldots, k$) gibt es genau $\binom{k}{l}$ Knoten.

3.3.2 Minimum- und Maximum-Halden

- Definition:
  - Ein *Maximum-Binomial-Baum* ist ein Binomial-Baum, der die Maximum-Bedingung (vgl. § 3.2.2) erfüllt.
  - Eine *Maximum-Binomial-Halde* ist eine (endliche) Folge von Maximum-Binomial-Bäumen, deren Grad streng monoton wächst.
  - *Minimum-Binomial-Baum* und *Minimum-Binomial-Halde* sind analog definiert.
- Folgerung:
  - Alle Binomial-Bäume innerhalb einer Binomial-Halde besitzen unterschiedlichen Grad.
  - Eine Binomial-Halde mit N Elementen besteht aus Binomial-Bäumen mit Graden $k_1 < \ldots < k_p$, sodass gilt: $N = \sum_{i=1}^{p} 2^{k_i}$, d. h. $k_1, \ldots, k_p$ sind genau die Ziffern mit Wert 1 in der Dualdarstellung von N.
  - Sowohl die Anzahl der Bäume in einer solchen Halde als auch deren Grade und Tiefen sind jeweils höchstens $O(\log_2 N)$.

3.3.3 Repräsentation von Binomial-Halden

- Jeder Knoten eines Binomial-Baums bzw. einer Binomial-Halde kann durch eine Datenstruktur mit folgenden Attributen repräsentiert werden:
  - `degree` speichert den Grad des Knotens.
  - `parent` verweist auf den Vorgänger des Knotens.
    (Bei einem Wurzelknoten ist `parent` gleich $\perp$.)
  - `child` verweist auf den Nachfolger des Knotens mit dem größten Grad.
    (Bei einem Blattknoten ist `child` gleich $\perp$.)
  - `sibling` verkettet alle Nachfolger eines Knotens in einer nach aufsteigendem Grad sortierten zirkulären Liste, das heißt:
    - Der Nachfolger mit dem größten Grad verweist auf den Nachfolger mit dem kleinsten Grad.
    - Jeder andere Nachfolger verweist auf den Nachfolger mit dem nächstgrößeren Grad.
    - Wenn es nur einen Nachfolger gibt, verweist er auf sich selbst.

    Außerdem verkettet `sibling` die Wurzelknoten aller Bäume der Halde in einer nach aufsteigendem Grad sortierten einfachen Liste, d. h. jeder Wurzelknoten verweist ggf. auf den Wurzelknoten mit dem nächstgrößeren Grad.
  - `entry` verweist auf das Objekt, das in diesem Knoten gespeichert werden soll und das seinerseits auf diesen Knoten zurückverweist (vgl. auch § 3.2.5).

- Die gesamte Halde wird durch einen Verweis auf den Wurzelknoten des Baums mit dem kleinsten Grad repräsentiert. (Bei einer leeren Halde ist dieser Verweis gleich $\perp$.)

Erläuterungen

- Mit den Attributen `child` und `sibling` können ohne Verwendung dynamischer Reihen o. ä. für jeden Knoten beliebig viele Nachfolger gespeichert werden.

- Die Gründe für die weiteren Details der Organisation (Reihenfolge der `sibling`-Verkettungen, Verwendung einfacher oder zyklischer Verkettung) werden in § 3.3.5 erläutert, nachdem in § 3.3.4 die Operationen auf dieser Datenstruktur beschrieben wurden.

3.3.4 Implementierung von Vorrangwarteschlangen

- Eine Minimum-Vorrangwarteschlange kann wie folgt durch eine Minimum-Binomial-Halde implementiert werden (und analog kann eine Maximum-Vorrangwarteschlange durch eine Maximum-Binomial-Halde implementiert werden).

Hilfsoperation

- `combine`: Zusammenfassen zweier Bäume B_1 und B_2 mit Grad k zu einem Baum mit Grad $k+1$

  1 Wenn die Priorität des Wurzelknotens von B_1 größer als die Priorität des Wurzelknotens von B_2 ist, mache B_1 zum Nachfolger mit dem größten Grad von B_2:

```
B2.sibling = nil
B2.degree = B2.degree + 1
B1.parent = B2
if B2.child == nil then
  B2.child = B1.sibling = B1
else
  B1.sibling = B2.child.sibling
  B2.child = B2.child.sibling = B1
end
```

  2 Andernfalls mache B_2 zum Nachfolger mit dem größten Grad von B_1.

Operationen

- `merge`: Vereinigen zweier Halden H_1 und H_2 zu einer neuen Halde H

  1. Erstelle eine leere Halde H
     und einen leeren Zwischenspeicher für bis zu drei Bäume.
  2. Setze $k = 0$.
  3. Solange H_1 oder H_2 oder der Zwischenspeicher nicht leer sind:
     1. Wenn H_1 nicht leer ist und sein erster (verbleibender) Baum Grad k besitzt, entnimm ihn aus H_1 und füge ihn zum Zwischenspeicher hinzu.
     2. Entsprechend für H_2.
     3. Wenn der Zwischenspeicher jetzt einen oder drei Bäume enthält, entnimm einen von ihnen und füge ihn am Ende von H an.
     4. Wenn der Zwischenspeicher jetzt noch zwei Bäume enthält, entnimm beide, fasse sie zu einem Baum mit Grad $k + 1$ zusammen (`combine`), und speichere ihn als „Übertrag“ für den nächsten Schritt im Zwischenspeicher.
     5. Erhöhe k um 1.

  Beispiel: H_1 mit $2^2 + 2^1 + 2^0 = 7$ Elementen und H_2 mit $2^3 + 2^2 + 2^1 = 14$ Elementen ergibt H mit $2^4 + 2^2 + 2^0 = 21$ Elementen

- `insert`: Einfügen eines Objekts mit einer bestimmten Priorität

  Erzeuge eine temporäre Halde mit einem einzigen Baum mit Grad 0, die das Objekt enthält, und vereinige sie mit der aktuellen Halde (`merge`).

- `minimum`: Auslesen eines Objekts mit minimaler Priorität

  1. Wenn die Halde leer ist: Fehler.
  2. Andernfalls durchsuche die Liste der Wurzelknoten nach einem Objekt mit minimaler Priorität und liefere es zurück.

- `extractMin`: Entnehmen eines Objekts mit minimaler Priorität

  1. Wenn die Halde leer ist: Fehler.
  2. Andernfalls durchsuche die Liste der Wurzelknoten nach einem Objekt mit minimaler Priorität und entferne diesen Knoten aus der Liste.
  3. Wenn dieser Knoten Nachfolger besitzt:
     Vereinige die Liste seiner Nachfolger (beginnend mit dem Nachfolger mit dem kleinsten Grad, der über `child` $\rightarrow$ `sibling` direkt zugreifbar ist)
     mit der verbleibenden Halde (`merge`).

- `changePrio`: Nachträgliches Ändern der Priorität eines Objekts
  1. Setze die Priorität des Objekts auf die neue Priorität.
  2. Wenn die neue Priorität kleiner oder gleich der alten ist:

     Solange die Priorität des Objekts kleiner als die seines Vorgängers ist:

     Vertausche das Objekt mit seinem Vorgänger
     durch „Über-Kreuz-Verbinden“ der beiden Objekte mit den zugehörigen Knoten.
  3. Andernfalls, sofern sich das Objekt nicht in einem Blattknoten befindet:

     Entferne das Objekt (`remove`)
     und füge es mit der neuen Priorität wieder ein (`insert`).

- `remove`: Entfernen eines Objekts
  1. Entnimm ein Objekt mit minimaler Priorität (`extractMin`).
  2. Wenn das entnommene Objekt nicht das zu entfernende Objekt ist:
     1. Verbinde das entnommene Objekt
        mit dem Knoten des zu entfernenden Objekts.
     2. Ändere die Priorität des entnommenen Objekts auf sich selbst (`changePrio`).

Anmerkung: Damit `remove` wie beschrieben funktioniert, ist es wichtig, dass `changePrio` Schritt 2 auch dann ausführt, wenn die neue Priorität gleich der alten ist.

Laufzeit der Operationen

- Wenn die Halde N Objekte enthält, besitzen alle `sibling`-Listen höchstens Länge $O(\log_2 N)$ und alle Bäume höchstens Tiefe $O(\log_2 N)$ (vgl. § 3.3.2).
- Daher besitzen alle Operationen höchstens Laufzeit $O(\log_2 N)$.

3.3.5 Entwurfsüberlegungen für die Datenstruktur

- Beim Vereinigen zweier Halden müssen ihre Bäume nach aufsteigendem Grad durchlaufen werden.
  Deshalb sind die Wurzelknoten in dieser Reihenfolge verkettet.

- Beim Entnehmen eines Wurzelknotens muss die Liste seiner Nachfolger mit der verbleibenden Wurzelliste vereinigt werden.
  Deshalb ist die Liste der Nachfolger eines Knotens ebenfalls nach aufsteigendem Grad sortiert verkettet.
  Dies weicht von der Darstellung in Cormen et al. (vgl. § 1.3) ab, wo die Liste nach absteigendem Grad verkettet ist und deshalb vor einer Vereinigung erst einmal invertiert werden muss.

- Beim Zusammenfassen zweier Bäume mit dem gleichen Grad wird einer der Bäume zum Nachfolger mit dem größten Grad des anderen Baums, d. h. er muss am Ende der Nachfolgerliste angehängt werden.
  Damit dies möglichst einfach und effizient geht, verweist `child` auf den Nachfolger mit dem größten Grad.

- Damit aber auch der Nachfolger mit dem kleinsten Grad direkt zugreifbar ist, ist die Liste der Nachfolger zirkulär verkettet.

- Für die Wurzelliste genügt jedoch eine einfache Verkettung.

3.4 Vergleich von binären und Binomial-Halden

- Die Laufzeit aller Operationen ist in beiden Implementierungen höchstens $O(\log_2 N)$.
- Bei der Implementierung mit binären Halden ist die Laufzeit der Operation „Auslesen“ sogar nur $O(1)$.
- Die Knoten von Binomial-Halden brauchen relativ viel Platz für Verwaltungsdaten (`degree`, `parent`, `child` und `sibling`), während binäre Halden nur die reinen Nutzdaten speichern.
- Binomial-Halden besitzen aber zwei Vorteile gegenüber binären Halden:
  - Ihre Kapazität ist prinzipiell nicht begrenzt.
  - Zwei Halden können auch effizient vereinigt werden.