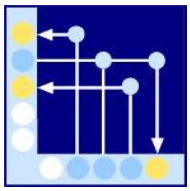
Hochschule Aalen
Fakultät Elektronik und Informatik
Studienbereich Informatik



Algorithmen und Datenstrukturen 2

Vorlesung im Wintersemester 2023/2024
Prof. Dr. habil. Christian Heinlein

2. Praktikumsaufgabe (28. November – 19. Dezember 2023)

Aufgabe 2: Vorrangwarteschlangen

Die Datei `binheap.h` auf der Vorlesungs-Webseite enthält eine unvollständige Klasse (bzw. Struktur) `BinHeap` als Schablone (template) mit zwei Typparametern `P` und `D`, die Minimum-Vorrangwarteschlangen mit Prioritäten des Typs `P` und zusätzlichen Daten des Typs `D` durch Binomial-Halden implementieren soll.

Implementieren Sie alle vorgegebenen Konstruktoren und Elementfunktionen dieser Klasse sowie die hierfür erforderlichen Datenstrukturen!

Beachten Sie unbedingt die in der Datei enthaltenen Kommentare, die das zu implementierende Verhalten spezifizieren! (Beispielsweise müssen die Nachfolger eines Knotens zirkulär verkettet und nach aufsteigendem Grad sortiert sein; `child` muss auf den Nachfolger mit dem größten Grad verweisen; usw.)

Die Datei enthält bereits Implementierungen der geschachtelten Hilfsklassen `Entry` und `Node`, die unverändert übernommen werden müssen. (Ausnahme: Es dürfen weitere Bestandteile hinzugefügt werden, obwohl dies zur Lösung der Praktikumsaufgabe nicht erforderlich ist.)

`Entry` ist eine interne Hilfsklasse zur Repräsentation von Einträgen mit Prioritäten des Typs `P` und zusätzlichen Daten des Typs `D`.

An der Stelle, an der `BinHeap<P, D>` für einen bestimmten Typ `P` verwendet wird, muss ein Kleiner-Operator (`operator<`) für den Typ `P` bekannt sein. Werte des Typs `P` dürfen in der Klasse `BinHeap` nur mit diesem Operator verglichen werden; die übrigen Vergleichsoperatoren (<=, >, >=, ==, !=) dürfen dort nicht verwendet werden.

Aus der Ausgabe von `dump` muss hervorgehen, wie die einzelnen Binomialbäume der Halde aufgebaut sind. Wenn in einen anfangs leeren `BinHeap<string, int>` nacheinander die Einträge `a 0` (d. h. ein Eintrag mit Priorität `a` und zusätzlichen Daten `0`) `b 1`, `c 2` usw. bis `k 10` eingefügt werden, muss `dump` exakt die folgende Ausgabe produzieren:

```
k 10
i 8
  j 9
a 0
  b 1
  c 2
    d 3
  e 4
    f 5
    g 6
      h 7
```

Verwenden Sie zur Implementierung der genannten Funktionen geeignete Hilfsfunktionen, insbesondere zum Zusammenfassen zweier Bäume und zum Vereinigen zweier Halden (vgl. die Beschreibung der Algorithmen in den Vorlesungsfolien)!

Alle Operationen außer `dump` dürfen höchstens logarithmische Laufzeit besitzen.

Testen Sie Ihre Implementierung sorgfältig und ausführlich! Auf der Vorlesungswebseite steht hierfür in der Datei `binheaptest.cxx` ein interaktives Testprogramm für die Klasse `BinHeap` zur Verfügung. (Ihre Implementierung muss aber natürlich auch mit anderen Typen als `string` und `int` für `P` und `D` funktionieren, sofern es für den Typ `P`, wie oben beschrieben, einen Kleiner-Operator gibt.)

Außerdem steht ein Shellskript `binheaptest.sh` zur Verfügung, das dieses Testprogramm mit einigen Befehlen aufruft und überprüft, ob die Ausgabe des Programms mit der erwarteten Ausgabe übereinstimmt.

Um automatisierte Tests der Implementierungen zu ermöglichen, dürfen die vorgegebenen Klassennamen und die Signaturen der Elementfunktionen nicht verändert werden, und es dürfen keine Diagnoseausgaben produziert werden.

Abgesehen von `<iostream>` (`cout`, `endl`, ...), dürfen keine Bestandteile der C++-Standardbibliothek oder anderer Bibliotheken verwendet werden.

Abzugeben ist die entsprechend erweiterte Datei `binheap.h`.
Die E-Mail mit der Abgabe muss als Betreff `Algo2 Gruppe NN` mit zweistelliger Gruppennummer `NN` (z. B. `05` oder `12`) und als Anhang die abzugebende Datei haben. Der Nachrichtentext muss für jedes Gruppenmitglied aus einer Zeile der Art `Vorname,Nachname,Matrikelnummer` bestehen.