Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2017/18
Feldmann / Foucard / Lichtblau / Streibelt / Krüger / Enghardt / Palmer

Aufgabenblatt 9

letzte Aktualisierung: 18. Januar, 17:57 Uhr

(3df3629496da8da334ed956cd3241d608d120faa)

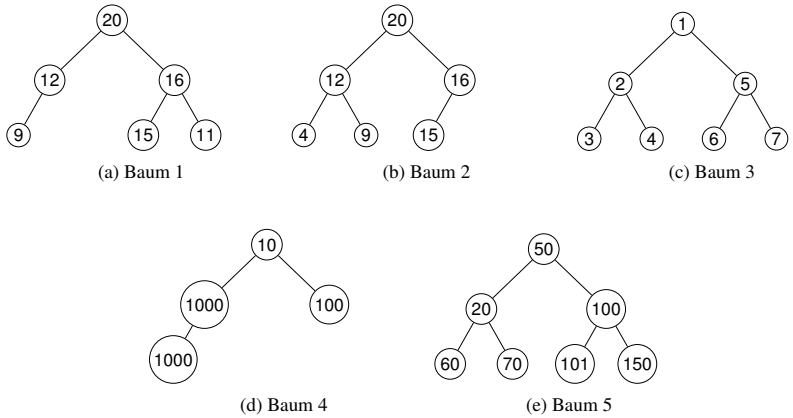
Ausgabe: Freitag, 19.01.2018
Abgabe: spätestens Mittwoch, 31.01.2018, 18:00

Thema: *Heaps*

Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des eecsIT mittels `gcc -std=c99 -Wall` kompilieren.
- Für die Hausaufgaben im Semester gibt es ein eigenes Subversion-Repository (SVN). Bitte die Hinweise auf Blatt 1 dazu beachten!
- Die Abgaben für Blatt 1 bis 4 erfolgen als Einzelabgaben. Ab Blatt 5 erfolgen Abgaben in festen Gruppen à 2 Personen.
- Die Gruppen werden vor Ausgabe von Blatt 5 gebildet. Solltet ihr keine Gruppe gebildet haben, werdet ihr einer Gruppe zugewiesen.
- Alle Abgaben müssen an der dafür vorgesehenen Stelle die Namen aller Gruppenmitglieder bzw. bei Einzelabgaben des Autors/der Autorin enthalten!
- Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Die finale Abgabe
  - für Gruppenabgaben erfolgt im Unterordner `Tutorien/t<xx>/Gruppen/g<xx>/Abgaben/Blatt<xx>/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Abgaben/Blatt<xx>/`

  WICHTIG: Die Abgabeordner werden immer von uns erstellt, sonst kommt es zu Konflikten! Benutzt zum Aktualisieren `svn up` im obersten Verzeichnis des Repositories!
- Benutzt für alle Abgaben die in der jeweiligen Aufgabe angegebenen Dateinamen (die von uns vorgegebenen Dateien haben das Wort ”vorgabe” im Dateinamen, du musst die Datei mit deiner Lösung also entsprechend umbenennen.)
- Gib bei den Programmieraufgaben den C-Quellcode ab und achte darauf, dass die Datei mit .c endet
- Bei Textaufgaben sind, wenn nicht anders angegeben, .txt Dateien zugelassen, die als Plaintext-Datei zu speichern sind. (Keine Word-Dateien umbenennen, etc.!)



(a) Baum 1 (b) Baum 2 (c) Baum 3

(d) Baum 4 (e) Baum 5

Abbildung 1: Heaps oder nicht?

1. Aufgabe: Binäre Heaps (1 Punkt)

1.1. Heap-Kriterien (0,5 Punkte) Betrachte die Bäume in der Abbildung 1 und gib jeweils an, ob es sich um einen Max-Heap, einen Min-Heap oder gar keinen Heap handelt.

Gibt weiterhin für die gültigen Heaps die Anordnung der Daten in der Arrayrepräsentation an.

Die Abgabe erfolgt über die Beantwortung der Fragen des Formulars `introprog_heap_identification_vorgabe.txt`, welches ihr im SVN findet.

1.2. Heap erstellen (0,5 Punkte) Wende die in der Vorlesung vorgestellte Funktion `build_heap` auf die angegebenen Arrays an und überführe diese in einen Max- bzw. Min-Heap.

a) Max-Heap: `[8,1,2,5,3,1]`
b) Min-Heap: `[42,23,15,16,8,4]`
c) Min-Heap: `[10,100,1,10,100,10]`

Die Abgabe erfolgt über die Beantwortung der Fragen des Formulars `introprog_build_heap_vorgabe.txt`, welches ihr im SVN findet.

Dabei wird der Zustand des Heaps wie folgt beschrieben:

- Jeder Knoten im Baum wird nummeriert.
- Die Nummerierung weist der Wurzel den Index 1, seinen Kindern die Werte 2 (links) und 3 (rechts) zu usw. (siehe Abbildung 3).
- Um zu beschreiben, dass der Knoten mit dem Index 5 den Wert 11 enthält, verwenden wir die Notation “5:11”.
- Die Angabe mehrerer Elemente erfolgt jeweils auf einer eigenen Zeile.
- Im Unterschied zu den binären Suchbäumen aus Aufgabenblatt 6 verwenden wir an dieser Stelle sowohl für die Indizes als auch für die Werte der Knoten Zahlen. Beachtet also, dass der erste Wert den Index repräsentiert und der zweite den Wert.

- Der Zustand des Baums aus Abbildung 2 wird somit wie folgt beschrieben:

```
1: 7
2: 8
3: 11
4: 12
5: 9
```

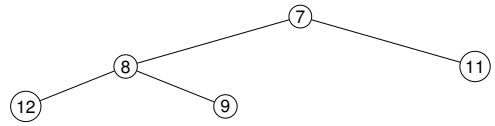


Abbildung 2: Min-Heap

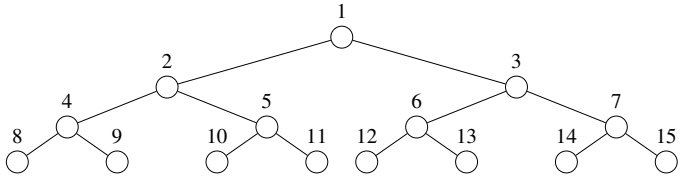


Abbildung 3: Nummerierung der Knoten im Baum

2. Aufgabe: Implementierung eines binären Max-Heaps (3 Punkte)

Eine Fluggesellschaft möchte das Einsteigen der Passagiere in ihre Flugzeuge beschleunigen. (Es wird angenommen, dass die Flugzeuge nur über den Eingang hinter dem Cockpit betreten werden.) Dazu sollen möglichst immer die Sitzreihen, welche sich am weitesten hinten im Flugzeug befinden, zuerst besetzt werden. (Die Sitzreihen werden vom Cockpit an hochgezählt.) Sobald ein Passagier am Gate eintrifft, wird er mit seiner Sitzreihe registriert. Neue Sitzreihen werden jeweils erst aufgerufen, wenn im Kabinengang wieder Platz vorhanden ist. Für diese Aufgabe soll ein System entwickelt werden.

Das System soll nach dem folgenden Muster funktionieren:

Eingabe <Zahl> Wenn ein Passagier beim Gate erscheint, soll dessen Sitzreihe (eine positive ganze Zahl) in die Warteliste eingetragen werden.

Eingabe 'n' Falls im Kabinengang des Flugzeugs wieder ausreichend Platz ist, soll das Bodenpersonal die Sitzreihe mit der größten Nummer angezeigt bekommen, für die ein Passagier am Gate wartet. Eine leere Warteliste wird durch die Ausgabe von "Leer" angezeigt.

Eingabe 'q' Das Programm wird durch Eingabe von "q" (für "quit") beendet.

Andere Eingaben Auf alle anderen Eingaben wird eine Fehlernachricht angezeigt.

Um die Möglichkeit von Betriebsfehlern zu reduzieren, müssen die folgenden Bedingungen erfüllt sein:

- Die Implementierung verwendet einen binären Max-Heap. Der Heap ist in `introprog_heap.h` als **`struct`** `heap` vorgegeben.
- Die Implementierung basiert auf dem Pseudocode der Vorlesung.
- Eventueller dynamischer Speicher muss freigegeben werden.

Die Ein-/Ausgabe sowie das Beenden des Programmes sind bereits vorgegeben.

Ein beispielhafter Aufruf des Programmes ist in Listing 1 dargestellt.

Listing 1: Programmbeispiel

```
1  > gcc -std=c99 -Wall introprog_maxheap.c introprog_main_maxheap.c \
2                        -o introprog_maxheap
3  > ./introprog_maxheap
4  [...]
5  > n
6  Leer!
7  > 12
8  > 33
9  > 3
10 > n
11 33
12 > n
13 12
14 > n
15 3
16 > n
17 Leer!
18 > foo
19 Fehler: Eingabe nicht erkannt!
20 > q
```

Die Codevorgabe im SVN besteht aus drei Dateien: `introprog_heap.h`, `introprog_main_maxheap.c` und `introprog_maxheap_vorgabe.c` (siehe Listing 2). Die letzte Datei soll angepasst werden und als `introprog_maxheap.c` eingereicht werden.

Folgende Funktionen müssen implementiert werden:

2.1. Speicherverwaltung (0,5 Punkte)

Implementiere anhand der Funktionssignaturen in der Vorgabe die Funktion `heap_create(` `↪ size_t capacity)` und `heap_free(heap* h)`. Achte dabei darauf, dass du nicht mehr Speicher als nötig reservierst, den Speicher korrekt initialisierst und am Ende alles wieder freigibst.

2.2. Funktion heapify (1 Punkte)

Schreibe die Funktion `void heapify(heap* h, int i)`, die sicherstellt, dass der Knoten i und seine Kinderknoten die Heap-Eigenschaft erfüllen, falls die Unterbäume der Kinderknoten die Heap-Eigenschaft erfüllen.

2.3. Funktion heap_insert (1 Punkte)

Schreibe die Funktion `int heap_insert(heap* h, int val)`, die ein neues Element in den Heap einfügt. Achte dabei darauf, dass die Datenstruktur auch nach dem Einfügen ein gültiger Heap ist. Orientiere dich an dem Pseudocode der Vorlesung.

Sollte der Funktion ein Heap übergeben werden, der schon vollständig aufgefüllt worden ist (d.h. ein Heap mit genau `capacity` Elementen), soll -1 zurückgeben werden.

2.4. Funktion heap_extract_max (0,5 Punkte)

Schreibe die Funktion `int heap_extract_max(heap* h)`, die das größte Element des Heaps zurückgibt und gleichzeitig vom Heap entfernt. Achte dabei darauf, dass auch nach dem Entfernen des Elements der Heap die Heapeigenschaft erfüllt. Orientiere dich an dem Pseudocode der Vorlesung.

Sollte der Funktion einen Heap übergeben werden, der leer ist (d.h. ein Heap mit 0 Elementen), soll -1 zurückgeben werden.

Hinweis: In bisherigen Aufgaben wurde der Einfachheit halber den Funktionen `malloc` und `calloc` ein **`int`** übergeben. Schaut man jedoch in die Dokumentation zu diesen Funktionen, wird dort die folgende Funktionssignatur gezeigt: **`void`** `*malloc(size_t size);`.

Der Datentyp `size_t` ist immer so groß wie ein Pointer (der verwendeten Architektur) und ist speziell für Pointerarithmetik gedacht. Auf der aktuell verbreitetsten Architektur x86/64 entspricht `size_t` einem **`unsigned long long`** (8 Byte). Damit macht es in der Praxis auf dieser Architektur keinen Unterschied, ob man (bei kleineren Datenmengen als 2GB) ein **`int`** `↪` oder `size_t` für den Aufruf und malloc/calloc oder zum Indizieren von Arrays verwendet. Ein Cast ist für die Konvertierung **`int`** `-> size_t` auch nicht erforderlich, da hierbei keine Informationen verloren gehen. Da C jedoch eine Sprache ist, die explizit dafür konzipiert wurde auf unterschiedlichen Plattformen programmiert werden zu können, wollen wir an dieser Stelle die Methode wählen, die auf allen Architekturen zuverlässig funktioniert.

Für die zu bearbeitende Aufgabe hat das zur Folge, dass wir den Funktionen `heap* heap_create(size_t capacity)` und **`void`** `heapify(heap* h,` `↪ size_t i)` nun kein **`int`**, sondern eine Variable vom Typ `size_t` übergeben. Im Umgang mit diesen Variablen ändert sich für euch nichts.

Listing 2: Codevorgabe `introprog_maxheap_vorgabe.c`

```c

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>

#include "introprog_heap.h"

/* Reserviere Speicher für einen Heap
 *
 * Der Heap soll dabei Platz für capacity Elemente bieten.
 *
 * Um konsistent mit den Parametern für malloc zu sein, ist capacity
 * als size_t (entspricht unsigned long auf x86/64) deklariert.
 * Da es sauberer ist, sind auch alle Indizes in dieser Aufgabe als
 * size_t deklariert. Ob man size_t oder int als Index für ein Array
 * verwendet, macht in der Praxis (auf x86/64) nur bei Arrays
 * mit mehr als 2.147.483.647 Elementen einen Unterschied.
 */
heap* heap_create(size_t capacity) {
}

/* Stellt die Heap Bedingung an Index i wieder her
 *
 * Vorraussetzung: Der linke und rechte Unterbaum von i
 * erfüllen die Heap-Bedingung bereits.
 */
void heapify(heap* h, size_t i) {
}

/* Holt einen Wert aus dem Heap
 *
 * Wenn der Heap nicht leer ist, wird die größte Zahl zurückgegeben.
 * Wenn der Heap leer ist, wird -1 zurückgegeben.
 */
int heap_extract_max(heap* h) {
}

/* Fügt einen Wert in den Heap ein
 *
 * Wenn der Heap bereits voll ist, wird -1 zurückgegeben,
 */
int heap_insert(heap* h, int key) {
}

/* Gibt den Speicher von einem Heap wieder frei
 */
void heap_free(heap* h) {
}
```