Praktikumstermin Nr. 09, INF
Dynamische Datenstrukturen - Binärer Suchbaum, Zeichenkette suchen mit C-Strings

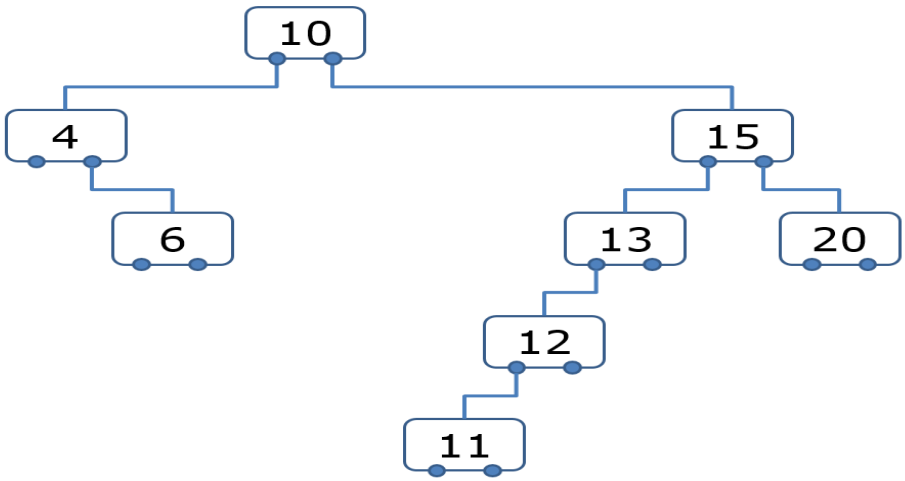
Abgabe im GIP-INF Praktikum der Woche 6.12.-10.12.2021.

2021-11-29: *Hinweis auf „Advent of Code" hinzugefügt*

2021-12-01: *Aufgabe für das Aufgaben-Tutorium hinzugefügt*

Aufgabe INF-09.01: Dynamische Datenstruktur: Binärer Suchbaum (duplikatfrei) über `int` Werten

Ein *Binärer Suchbaum (ohne Duplikate) über* `int` *Werten* ist eine Datenstruktur, in der `int` Werte in den Knoten der Datenstruktur nach den im folgenden beschriebenen Regeln gespeichert werden.



Jeder Knoten der Datenstruktur speichert genau einen `int` Wert und besitzt genau einen *Elternknoten* (Ausnahme: *Wurzelknoten* des Baums, der keinen Elternknoten besitzt) und höchstens zwei *Kindknoten*.

Der *erste* in den Baum einzufügende `int` Wert wird im neu zu erzeugenden Wurzelknoten des Baums abgelegt.

Jeder weitere einzufügende `int` Wert wird nach folgendem Prinzip in den Baum eingefügt: Ausgehend vom Wurzelknoten wird der neue Wert mit dem im jeweiligen Knoten gespeicherten Wert verglichen.

1. Ist der neue Wert *gleich* dem Wert im Knoten, so wird der neue Wert nicht erneut in den Baum eingefügt (*duplikatfreier* Baum).

2. Ist der neue Wert *kleiner* dem Wert im Knoten und besitzt der Knoten *keinen* linken Kindknoten, so wird der neue Wert in einen neu zu erzeugenden linken Kindknoten eingefügt.

3. Ist der neue Wert *kleiner* dem Wert im Knoten und besitzt der Knoten einen linken Kindknoten, so wird die Prüfung ab Fall 1. für den linken Kindknoten erneut vorgenommen.

   Fälle 4. und 5. sind analog zu 2. und 3.:

4. Ist der neue Wert *größer* dem Wert im Knoten und besitzt der Knoten *keinen* rechten Kindknoten, so wird der neue Wert in einen neu zu erzeugenden rechten Kindknoten eingefügt.

5. Ist der neue Wert *größer* dem Wert im Knoten und besitzt der Knoten einen rechten Kindknoten, so wird die Prüfung ab Fall 1. für den rechten Kindknoten erneut vorgenommen.

Programmieren Sie eine geeignete Datenstruktur `BaumKnoten` sowie Funktionen `einfuegen()` und `ausgeben()`, um einen duplikatfreien Binärbaum über `int` Werten gemäß den Testläufen zu realisieren.

Welche Komponenten die Datenstruktur `BaumKnoten` enthalten muss, sollen Sie anhand der Notwendigkeiten der Funktionen `einfuegen()` und `ausgeben()` entscheiden.

Verwenden Sie keine globalen oder `static` Variablen.

Deklarieren Sie die Datenstruktur `BaumKnoten` sowie die Funktionsprototypen für `einfuegen()` und `ausgeben()` in einer Headerdatei `binaerer_suchbaum.h`, und innerhalb eines Namespaces `suchbaum`.

Implementieren Sie die Funktionen `einfuegen()` und `ausgeben()` in einer Datei `binaerer_suchbaum.cpp`. Sie können gerne zusätzliche

Hilfsfunktionen definieren, dann aber innerhalb des Namespaces `suchbaum`.
Die Ausgabefunktion `ausgeben()` rückt die Knotenwerte entsprechend ihrer Tiefe im Baum (d.h. Abstand vom Wurzelknoten) ein, mit zwei Pluszeichen pro Tiefenstufe. Der Baum ist bei der textuellen Ausgabe „um 90 Grad gegen den Uhrzeigersinn gedreht" im Vergleich zur Diagrammdarstellung.
D.h. zu einem Baumknoten wird erst der rechte Teilbaum ausgegeben, dann der Wert des Knotens selbst, dann der linke Teilbaum.
Wegen der Selbstähnlichkeit (Teilbaum sieht von der Struktur aus wie der gesamte Baum): Realisieren Sie die Ausgabe über eine rekursive Funktion

```
void suchbaum::knoten_ausgeben(BaumKnoten* knoten, int tiefe);
```

… die aus der Funktion `ausgeben()` aufgerufen wird und genau das obige Ausgabeprinzip umsetzt (lassen Sie sich von der „Türme von Hanoi" Funktion inspirieren, falls nötig…).

Legen Sie im Projekt eine leere Headerdatei `catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei.

Legen Sie im Projekt eine Datei `test_binaerer_suchbaum.cpp` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Datei.

Implementieren Sie die `main()` Funktion in einer Datei `suchbaum_main.cpp`.

Testläufe (Benutzereingaben sind unterstrichen):

```
===============================================================================
All tests passed (71 assertions in 7 test cases)

Leerer Baum.
Naechster Wert (99 beendet): ? 10
Naechster Wert (99 beendet): ? 4
Naechster Wert (99 beendet): ? 6
Naechster Wert (99 beendet): ? 15
Naechster Wert (99 beendet): ? 13
Naechster Wert (99 beendet): ? 12
Naechster Wert (99 beendet): ? 15
Naechster Wert (99 beendet): ? 20
```

```
Naechster Wert (99 beendet): ? 11
Naechster Wert (99 beendet): ? 15
Naechster Wert (99 beendet): ? 99
++++20
++15
++++13
++++++12
++++++++11
10
++++6
++4
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (71 assertions in 7 test cases)

Leerer Baum.
Naechster Wert (99 beendet): ? 3
Naechster Wert (99 beendet): ? 3
Naechster Wert (99 beendet): ? 3
Naechster Wert (99 beendet): ? 2
Naechster Wert (99 beendet): ? 99
3
++2
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (71 assertions in 7 test cases)

Leerer Baum.
Naechster Wert (99 beendet): ? 99
Leerer Baum.
Drücken Sie eine beliebige Taste . . .
```

```cpp
#define CATCH_CONFIG_RUNNER
#include "catch.h"

#include <iostream>
using namespace std;

#include "binaerer_suchbaum.h"


int main()
{
    // Aufruf der Unit Tests ...
    Catch::Session().run();
```

Aufgabe INF-09.02:
Zeichenkette suchen, mit C-Strings, iterativ

Schreiben Sie eine Headerdatei `zeichenkette_suchen.h` sowie eine `.cpp` Datei `zeichenkette_suchen.cpp` für eine C++ Funktion

```
int zeichenkette_suchen(const char* text, const char* zkette);
```

welche ermittelt, ob die Zeichenkette `zkette` in dem einzeiligen Text `text` (ggfs. mit Leerzeichen und/oder Satzzeichen) vorkommt.
Die maximale Textlänge von `text` und `zkette` betrage jeweils `20` vom Benutzer eingegebene Zeichen.
Die zu suchende Zeichenkette `zkette` sei nicht leer, d.h. enthalte außer dem Nullterminator noch mindestens ein Zeichen.
Der zu durchsuchende Text darf aber leer sein.
Der Benutzer mache nur korrekte Eingaben, d.h. ihr Programm kann davon ausgehen, dass z.B. die eingegebenen Zeichenketten nicht länger als 20 eingegebene Zeichen sind.

Sollte die Zeichenkette nicht in dem Text vorkommen, so soll der Wert `-1` zurückgegeben werden. Anderenfalls wird die Startposition zurückgegeben, ab der das erste (linkeste) Vorkommen von `zkette` in `text` beginnt. Die Zählung der Positionen im Text beginne bei `0`.

Ihre `zeichenkette_suchen()` Funktion **muss iterativ arbeiten**, d.h. muss **Schleifen verwenden** und **darf *nicht rekursiv* arbeiten**!

*Die Funktion darf keinerlei C-String Funktionen aus Libraries wie der `cstring` Library oder C++ `string` benutzen, sondern ausschließlich über den Array-Indexoperator auf die einzelnen Zeichen der beiden C-Strings zugreifen. Sollten Sie für ihre Programmierung die Länge eines C-Strings benötigen, so müssen sie auch dafür eine eigene Hilfsfunktion programmieren, die nur den Array-Indexoperator für den Zugriff auf die einzelnen Zeichen des C-Strings benutzt. Beachten Sie, dass C-Strings immer mit dem Null-Terminator enden und dass durch Vergleich mit `'\0'` das Ende eines C-Strings erkannt werden kann. Beachten Sie ferner, dass die Funktion `zeichenkette_suchen()` ihre **Parameter als Pointer auf konstante Character entgegennimmt**. D.h. auch alle **Hilfsfunktionen**, die von dort aus aufgerufen werden (z.B. `my_strlen()`), **müssen dieses „Versprechen" ebenfalls einhalten**, falls sie einen oder beide der Parameter weitergereicht bekommen und diese Pointer daher auch **als Pointer auf konstante Character behandeln**!*

Legen Sie im Projekt eine leere Headerdatei `catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei.

Legen Sie im Projekt eine Datei `test_zeichenkette_suchen.cpp` an mit folgendem Inhalt (liegt auch in Ilias vor):

```
// Datei: unit_tests.cpp

#include "catch.h"
#include "suchen.h"

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
groesser 1") {
    REQUIRE(zeichenkette_suchen("abcdabcde", "cda") == 2);
    REQUIRE(zeichenkette_suchen("abcdabcde", "de") == 7);
    REQUIRE(zeichenkette_suchen("abcdabcde", "dex") == -1);
    REQUIRE(zeichenkette_suchen("abcdabcde", "xyz") == -1);
    REQUIRE(zeichenkette_suchen("abcdabcde", "abcdabcd") == 0);
    REQUIRE(zeichenkette_suchen("abcdabcde", "abcdabcdx") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
1") {
    REQUIRE(zeichenkette_suchen("abcdabcde", "a") == 0);
    REQUIRE(zeichenkette_suchen("abcdabcde", "c") == 2);
    REQUIRE(zeichenkette_suchen("abcdabcde", "e") == 8);
    REQUIRE(zeichenkette_suchen("abcdabcde", "x") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 1") {
    REQUIRE(zeichenkette_suchen("a", "a") == 0);
    REQUIRE(zeichenkette_suchen("a", "c") == -1);
    REQUIRE(zeichenkette_suchen("a", "xy") == -1);
    REQUIRE(zeichenkette_suchen("a", "aa") == -1);
}

TEST_CASE("Zeichenkette suchen, leerer Text") {
    REQUIRE(zeichenkette_suchen("", "a") == -1);
    REQUIRE(zeichenkette_suchen("", "abc") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 20 Zeichen") {
    REQUIRE(zeichenkette_suchen("abcdefghij1234567890", "90") == 18);
    REQUIRE(zeichenkette_suchen("12345678901234567890", "90") == 8);
    REQUIRE(zeichenkette_suchen("abcdefghij1234567890", "9012") == -1);
}

// Vorgegebene Testläufe müssen selbst als Testcases programmiert werden ...
TEST_CASE("Vorgegebene Testlaeufe") {

}
```

Ergänzen Sie die Unit Tests in dieser Datei um die Testfälle, die in den unten angegebenen Testläufen vorgegeben sind.

Schreiben Sie ferner ein C++ Hauptprogramm, welches die Eingaben entgegennimmt, die Funktion aufruft und das Ergebnis ausgibt (siehe Testläufe). Ergänzen Sie dabei den folgenden Programmrahmen:

```
// Datei: zeichenkette_suchen_main.cpp

#define CATCH_CONFIG_RUNNER
#include "catch.h"

#include <iostream>
using namespace std;

#include "zeichenkette_suchen.h"

int main()
{
    Catch::Session().run();

    // Ihr Code ab hier ...


    system("PAUSE");
    return 0;
}
```

Testläufe: *(Benutzereingaben sind zur Verdeutlichung unterstrichen)*

```
===============================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: bcd99
Die Zeichenkette 'bcd99' ist NICHT in dem Text 'abcdefg' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: efg
Die Zeichenkette 'efg' ist in dem Text 'abcdefg' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: abc
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
```

```
Die Zeichenkette 'abcde' ist NICHT in dem Text 'abc' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
==========================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: 012 abc abc 89
Bitte geben Sie die zu suchende Zeichenkette ein: abc
Die Zeichenkette 'abc' ist in dem Text '012 abc abc 89' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
==========================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: xy abc abcdefgh
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
Die Zeichenkette 'abcde' ist in dem Text 'xy abc abcdefgh'
enthalten.
Sie startet ab Zeichen 7 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
==========================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: xyz 123 456 abc
Bitte geben Sie die zu suchende Zeichenkette ein: 123 4
Die Zeichenkette '123 4' ist in dem Text 'xyz 123 456 abc'
enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
==========================================================================
All tests passed (26 assertions in 6 test cases)

Bitte geben Sie den Text ein: abc defg
Bitte geben Sie die zu suchende Zeichenkette ein: abc d
Die Zeichenkette 'abc d' ist in dem Text 'abc defg' enthalten.
Sie startet ab Zeichen 0 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

Für das Aufgaben-Tutorium am Freitag 3.12.2021, als freiwillige Aufgabe:

(Freiwillige) Aufgabe INF-09.03: Spiel „Regnende Kästchen“

Programmieren Sie ein einfaches „Computerspiel“ gemäß den folgenden Anforderungen, unter Benutzung der `CImg` Library.

Bei dem Spiel „regnen“ Kästchen von oben nach unten. Durch Anklicken der Kästchen werden diese wieder nach oben zurückgesetzt und regnen dann mit leicht erhöhter Geschwindigkeit erneut nieder. Das Spiel endet, wenn ein Kästchen die untere Kante des Fensters überschritten hat.

Achten Sie bei Ihrer Lösung darauf, dass die Kästchen auch wirklich an den oberen Rand des Fensters zurückgesetzt werden. D.h. die Oberkante des Kästchens sollte beim Zurücksetzen an den oberen Rand des Fensters stoßen, es sollte zuerst einmal keine Lücke sichtbar sein, bevor das Kästchen erneut „herunterfällt“.

Legen Sie ein neues leeres Projekt an. Legen Sie dort eine neue Headerdatei `CImgGIP05.h` an und copy-pasten Sie den Inhalt der entsprechenden Datei aus Ilias in diese Datei. Legen Sie ferner eine Quellcodedatei (`.cpp` Datei) mit einem von Ihnen frei wählbaren Namen an und copy-pasten Sie den gegebenen Coderahmen in diese Datei. Komplettieren Sie dann den Coderahmen.

```
#include <iostream>
#define CIMGGIP_MAIN
#include "CImgGIP06.h"
using namespace std;

struct Box
{
    int x;
    int y;
    int delta_y; // aktuelle Fallgeschwindigkeit dieses Kaestchens
};

const int box_max = 10, box_size = 40;

// draw_boxes():
// Die Anzahl der Kaestchen wird nicht als zweiter Parameter uebergeben,
// da diese Anzahl als globale Konstante box_max im gesamten Programm bekannt ist ...
void draw_boxes(Box boxes[])
{
    // Loesche den bisherigen Fensterinhalt komplett,
    // d.h. komplettes Neuzeichnen aller Kaestchen etc ...
    gip_white_background();
```

```
    // Und jetzt alle Kaestchen zeichnen mittels gip_draw_rectangle().
    // Linke obere Ecke: (boxes[i].x, boxes[i].y)
    // Groesse: box_size x box_size
    // Farbe: blue

    // TO DO

}

// update_boxes():
// 1. Berechne neue y-Koordinate (jedes Kaestchen boxes[i] faellt um
boxes[i].delta_y)
// 2. Pruefe, ob ein Kaestchen das untere Fensterende gip_win_size_y (ist ein int)
//    ueberschritten hat. Falls ja: gib sofort false zurueck
// 3. (Sonst:) gib true zurueck, wenn keines der Kaestchen den unteren Rand
//             ueberschritten hatte
// Die Anzahl der Kaestchen wird nicht als zweiter Parameter uebergeben,
// da diese Anzahl als globale Konstante box_max im gesamten Programm bekannt ist ...
bool update_boxes(Box boxes[])
{
    // TO DO
}

// handle_mouse_click():
// Annahme: Funktion wird nur aufgerufen, wenn die Maus wirklich
// geklickt wurde. Die Funktion braucht dies also nicht mehr zu pruefen.
//
// Nimm die Koordinaten der Mausposition und pruefe dann fuer jedes Kaestchen,
// ob die Koordinaten innerhalb des Kaestchens liegen.
// Falls ja:
// * Erhoehe die Fallgeschwindigkeit des Kaestchens um 10
// * Setze das Kaestchen an den oberen Rand
void handle_mouse_click(Box boxes[])
{
    int mouse_x = gip_mouse_x();
    int mouse_y = gip_mouse_y();

    // TO DO
}

int main()
{
    Box boxes[box_max] = { 0 };
    bool keep_going = true;

    // Initialisiere alle Kaestchen ...
    for (int i = 0; i < box_max; i++)
    {
        // Die "+10" bewirken, dass alle Kaestchen leicht nach rechts versetzt werden
        // und somit links vom linksten Kaestchen noch etwas Platz bleibt (dieses
        // also nicht am Rand klebt).
        // Die "+20" sorgen fuer einen Zwischenraum von 20 zwischen den Kaestchen ...
        boxes[i].x = i * (box_size + 20) + 10;
        boxes[i].y = 0;
        // Startgeschwindigkeit ist ganzzahlige Zufallszahl zwischen 10 und 30 ...
        boxes[i].delta_y = gip_random(10, 30);
```

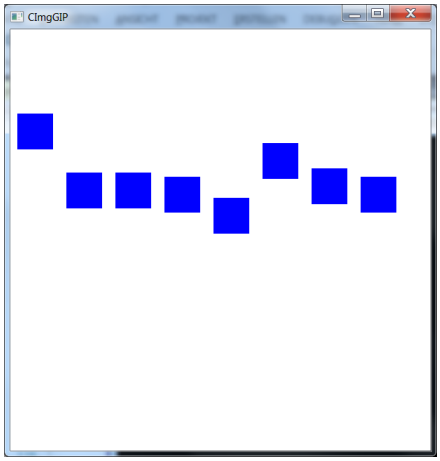
```
    }

    while (keep_going && gip_window_not_closed())
    {
        draw_boxes(boxes);

        for (int loop_count = 0; loop_count < 200; loop_count++)
        {
            gip_wait(5); // warte 5 Milli-Sekunden
            if (gip_mouse_button1_pressed())
            {
                handle_mouse_click(boxes);
            }
        }
        // Berechne die neue Fall-Position aller Kaestchen und pruefe,
        // ob eines der Kaestchen unten aus dem Fenster "herausgefallen" ist
        // (falls ja, wird der Wert false zurueckgegeben) ...
        keep_going = update_boxes(boxes);
    }

    return 0;
}
```

Testlauf:



Hinweis: Advent of Code

Wenn Sie gerne weiteres, interessantes "Programmierfutter" hätten, dann möchte ich Sie auf den "Adventskalender" *Advent of Code* hinweisen, der in den letzten Jahren und auch dieses Jahr zur Adventszeit veröffentlicht wurde/wird: https://adventofcode.com/. Dort wird wie beim Adventskalender üblich jeden Tag eine neue Programmieraufgabe veröffentlicht, die man in einer beliebigen Programmiersprache (also

warum nicht auch in C++) lösen soll. Die Aufgaben bleiben auch nach der Adventszeit verfügbar und auch die Aufgaben der letzten Jahre sind immer noch verfügbar, z.B. https://adventofcode.com/2015 …

Man muss sich auf dieser Webseite allerdings anmelden, da jeder Teilnehmer eine spezifisch für ihn variierte Eingabedaten bekommt mit entsprechend spezifischen Ergebnisdaten, die dann zur Prüfung hochgeladen werden können. Der Code des eigenen Lösungsprogramms kann und soll nicht hochgeladen werden (reine "Black Box" Prüfung, wie das auch der Jenkins macht, der schaut auch nicht in ihren hochgeladenen Code, sondern prüft nur die Eingaben/Ausgaben ihres Programms …)

Der *Advent of Code* ist "eigentlich" ein Wettbewerb, bei dem die ersten 100 hochgeladenen korrekten Lösungen Punkte bekommen und es entsprechend ein "Leaderboard" gibt. Aber wenn man "nur" das Ziel hat, eine Lösung zu finden und durch Hochladen der Ergebnisdaten die Korrektheit prüfen zu lassen, dann ist das ein Vorgehen in gewisser Weise ähnlich zu unseren Offline-Aufgaben (auch wenn Sie "bei uns" ihren Programmcode hochladen, bei *Advent of Code* nicht…).

Die Aufgaben beim *Advent of Code* sind allerdings deutlich komplexer als das, was wir im Rahmen der GIP Veranstaltung an Aufgaben behandeln und die Aufgaben setzen eine Programmiererfahrung voraus, die weit über das hinausgeht, was im Rahmen des GIP Moduls als Lernziel angesetzt ist. Auch wird man für eine "elegante" Lösung einiger Aufgaben ggfs. dynamische Datenstrukturen verwenden, die wir erst später (bald, noch im Dezember) in diesem Semester behandeln werden.

Wenn Sie also "Programmier-Herausforderungen suchen", dann wäre das eine Möglichkeit …

Aber: Die Inhalte des Studiums sollten aus meiner Sicht Priorität bei Ihnen haben, denn das ist ihr aktueller Beruf. Beim Advent of Code mitzumachen ist dann zwar für GIP "berufsunterstützend", aber wenn es sinnvoller wäre, Zeit in andere Studienfächer zu investieren, dann tun Sie dies bitte zu ihrem eigenen Besten… Die Advent of Code Aufgaben können Sie dann ja später immer noch bearbeiten (diese bleiben auch jahrelang nach der Adventszeit verfügbar) …