


Artemis 6.4.0

Course Overview

  ge64baw

Courses > Praktikum: Grundlagen der Programmierung WS22/23 > Exercises > W10H01 - Pengu Trials

W10H01 - Pengu Trials

Hausaufgabe

Easy

Submission due:

7 months ago

Points: 8 of 8

Assessment: automatic ?

Complaint due:

7 months ago

 100%  (7 months ago)

GRADED

Recent results:



Show all results

Tasks:

Pengu Trials

Pinguine sind unglaublich disziplinierte Lebewesen. Einige Pinguine wollen jedoch über sich selbst hinauswachsen und die besten Pingus aller Zeiten werden. Sie streben nach einer Mitgliedschaft in der *School of Penguins*. Der Ruhm lockt viele kleine Pinguine an, doch nur die wenigsten überstehen die harten Vorbereitungen. Von den wenigen, die diese Phase tapfer gemeistert haben, schaffen es noch viel weniger, die finalen Aufnahmeprüfungen zu überleben. (*Dramatische Soundeffekte: BLITZ, DONNER, STURM!*).

Die Prüfungen der School of Penguins werden Trials genannt:

1. **Trials of the Grasses:**  
Mitglieder müssen sich mit ihrer Umwelt auskennen. Besser als jedes andere Lebewesen. Sie müssen die Welt der Pflanzen nicht nur erkunden und erkennen können, sondern das Innere der Pflanzen selbst verstehen. Von den Wurzeln, über die Äste, bis hin zu den kleinsten Blättern.
2. **Trials of the Sword:**  
Ein Pinguin muss schnell und elegant mit einem Schwert hantieren können. Um diese Fähigkeit zu beweisen, müssen sie meist hoch dimensionale Datenstrukturen mit ihrem Schwert bearbeiten, ohne sich dabei selbst zu verletzen. Eine wahrlich gefährliche Challenge.
3. **Trials of the Mountains:**  
Die Pinguine haben alles über den Kontinent gelernt. Dazu gehören auch die Berglandschaften und die Kreaturen, die sich dort aufhalten. Dieses Wissen ist essenziell für das Reisen durch die Berge, ohne dabei unerwartet auf Monster zu treffen.
4. **Trials of the Dreams:**  
Die letzte und schwierigste Prüfung. Um die geistige Exzellenz des Prüflings zu testen. Ein Mitglied der School of Penguins muss im Schlaf Schlösser knacken können. Eine wichtige Fähigkeit, um sich durch verschlossene Gänge hindurch kämpfen zu können und beim entschlüsseln komplexer Verriegelungen noch einmal Ruhe und Energie aufzuladen.

Jetzt bist du an der Reihe. Schaffst du die Pengu Trials um das Pengu-Medallion als Mitglied der School of Penguins tragen zu dürfen?


Story

Durch deine Prüfungen begleitet dich deine treue Freundin und Zauberin Eve von Cintra: 1.

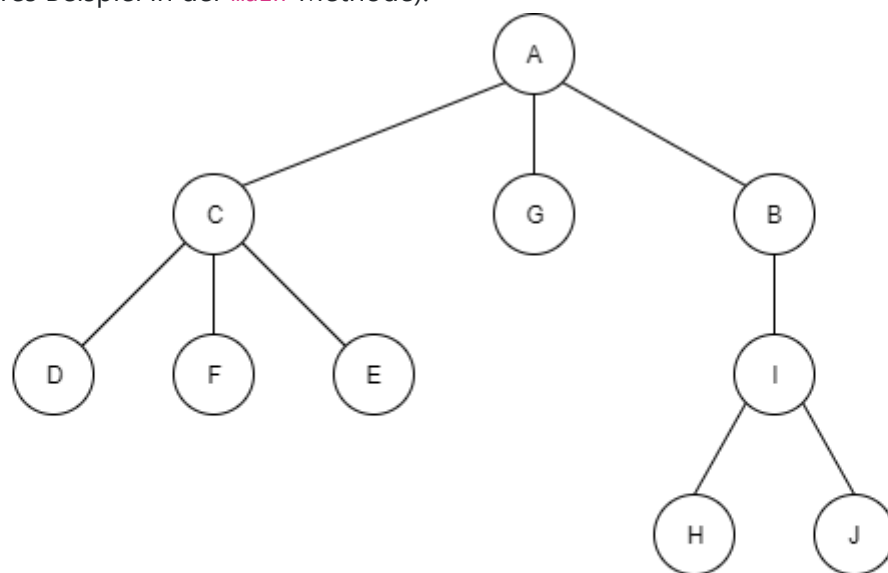


Pergament 0 of 1 tests passing

Aufgaben

1.  **Trials of the Grasses** 1 of 1 tests passing
- In `TrialOfTheGrasses.java` findest du die Klasse `TreeNode`, die einen Baum modelliert. Ein `TreeNode` speichert seinen Wert in `label` und hat beliebig viele Kindknoten (`nodes`). Implementiere die Methode `flatten()`, die einen `Stream` bestehend aus allen `TreeNodes` des Baums in preorder zurückgeben soll (nicht den `labels` sondern den Knoten selbst). Preorder bedeutet dabei, dass zunächst der Knoten selbst und anschließend die

entsprechenden **Streams** der Kinder traversiert werden sollen.  
 Hier ein kleines Beispiel (Weiteres Beispiel in der **main**-Methode):



## 2. **Trials of the Sword** 1 of 1 tests passing

In **TrialOfTheSword.java** findest du die Klasse **FlatArray**, die mehrdimensionale Arrays speichern können soll. Implementiere die Methode **computeIndex(int... idxs)**, die die Position des durch die Argumente beschriebenen Elements im **FlatArray** bestimmt. Im Folgenden wird dir die innere Struktur des **FlatArray**s erklärt. Zusätzlich bietet die Klasse bereits eine Menge von Methoden, die in den Kommentaren erklärt werden. Ein **FlatArray** soll mehrdimensionale Arrays (Arrays aus Arrays) in einem normalen Array (**T[] values**) speichern. In dem **int** array **dims** werden die Längen der Arrays gespeichert.

Angenommen wir haben ein zweidimensionales Array von **As** (eine beliebige Klasse):  $A[3][5] = \begin{Bmatrix} \{a_{00}, a_{01}, a_{02}, a_{03}, a_{04}\} \\ \{a_{10}, a_{11}, a_{12}, a_{13}, a_{14}\} \\ \{a_{20}, a_{21}, a_{22}, a_{23}, a_{24}\} \end{Bmatrix}$

Solch ein Array sollte in **FlatArray** wie folgt dargestellt werden: **dims** = {3, 5} und **values** = { $a_{00}, a_{01}, a_{02}, a_{03}, a_{04}, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{20}, a_{21}, a_{22}, a_{23}, a_{24}$ }

Die Elemente der 3 äußeren Arrays werden also einfach hintereinander in einem einzelnen Array gespeichert. Der Aufruf der Methode **computeIndex(1,3)** sollte dementsprechend den Index 8 zurückgeben, da  $a_{13}$  in **values[8]** liegt.

Hätten wir nun ein Array mit **dims** = {3, 5,  $x$ }, würde in **values** jedes  $a_{ij}$  durch  $x$  Elemente ersetzt werden (die Werte der innersten Arrays). Dies lässt sich beliebig-dimensional fortsetzen.

## 3. **Trials of the Mountains** 1 of 1 tests passing

In **TrialOfTheMountains.java** findest du die Klasse **Mountain**. Dabei handelt es sich um eine Klasse, die Netzwerke von Bergen modelliert. Ein Berg ist genau von einer Monsterspezies bewohnt (**beast**), jedoch können zwei benachbarte Berge (**neighbours**) nie von der gleichen Spezies besetzt sein. Implementiere die Methode **assignBeasts(Mountain[])**, die jedem Berg des gegebenen Graphen eine Spezies zuweist. Die Anzahl der verwendeten Spezies muss dabei nicht minimal sein, Lösungen, die jedem Mountain eine andere Spezies zuweisen, werden jedoch nicht akzeptiert.  
**Hinweis:** Du kannst davon ausgehen, dass gilt: "Wenn Mountain A Nachbar ist von B, dann ist auch B ein Nachbar von A" und das Attribut **Beast** **beast** eines Mountains ist zu Beginn immer **null**.

**Tipp:** Enums bieten die Methoden **values()** und **ordinal()** an. Sieh dir diese in den Java Docs an, sie könnten hilfreich sein.

## 4. **Trials of the Dreams** 1 of 1 tests passing

In **TrialOfTheDreams.java** findest du die Methoden **lockPick**, die dazu benutzt werden können, ein Schloss zu knacken. Ein Schloss (**lock**) ist eine Funktion, die als Eingabe ein **byte** Array erwartet und **true** zurückgibt, wenn das Array eine passende Kombination ist, um das Schloss zu öffnen (andernfalls **false**). Die Methode **lockPick(Function<byte[], Boolean>)** ist bereits vorgegeben. Sie nutzt ein bekanntes Verfahren "iterative deepening search", um nach einer Kombination zu suchen. Dazu wird iterativ die mögliche maximale Länge des Schlüssels erhöht. Damit diese Methode funktioniert, musst du noch die beiden Hilfsmethoden **lockPick(Function<byte[], Boolean>, int)** und **lockPick(Function<Byte[], Boolean>, List<Byte>, int)** implementieren.

**lockPick(Function<byte[], Boolean>, int):**

Diese Methode soll die gegebene **Function<byte[], Boolean>** in eine **Function<Byte[], Boolean>** "umwandeln" (**wrapper**), um eine einfachere Implementierung anfertigen zu können. Hast du den **wrapper** erstellt, kannst du **lockPick(Function<Byte[], Boolean>, List<Byte>, int)** dazu nutzen, die Lösung berechnen zu lassen. Achte darauf, dass der Rückgabotyp von **lockPick(Function<Byte[], Boolean>, List<Byte>, int)** eine **List<Byte>** ist. Wurde keine Kombination gefunden, soll **null** zurückgegeben werden, ansonsten eine **Byte**-Liste mit den entsprechenden Werten, welche das Schloss knacken.


**lockPick(Function<Byte[], Boolean>, List<Byte>, int):**

Diese Methode löst das eigentliche Problem rekursiv für Kombinationen mit Länge  $\leq$  **maxlen**. Existiert keine Kombination, die das Schloss öffnet, soll **null** zurückgegeben werden. **Tipp:** Für diese Aufgabe reicht ein einfacher Brute-Force Approach (naiv alle Möglichkeiten ausprobieren).

## Tests

### Exercise details

Release date:	Jan 5, 2023 18:30
Start date:	Jan 5, 2023 18:30
Submission due:	Jan 22, 2023 18:00
Complaint due:	Jan 29, 2023 18:00

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have **998** complaints left. 

[About](#)

[Request change](#)   [Release notes](#)   [Privacy Statement](#)   [Imprint](#)