Induktion: “Türme von Hanoi” ohne Rekursion und Minimax

Mathilde Rüfenacht

1 Kontext

Diese Unterrichtssequenz enthält 2 Teile, insgesamt 4 bis 6 Lektion Einheiten einer 10. Klasse mit Akzentfach Mathe entsprechen. Diese Sequenz ist für eine Vertiefung des Konzepts der Induktion gemeint. Die SuS müssen das Konzept von Induktion schon in einfachen Beispielen gesehen haben. Sie müssen auch schon gut mit Listen arbeiten können.

2 Einführung

2.1 Ziel

Das Ziel dieser Sequenz ist die Vertiefung des Konzepts der Induktion in verschiedenen Kontexten und die Vorbereitung auf die Rekursion. Die Induktion wird auf verschiedene Weisen von einfachen bis fortgeschrittenen Aufgaben eingesetzt. Die Aufgaben werden auch zu Programmen führen, in welchen die Induktionsformeln ohne Rekursion implementiert werden.

2.2 Konzept der Sequenz

In Teil 1 wird die Induktion angewendet werden, um das Spiel “Turme von Hanoi” ohne Rekursion zu programmieren.

In Teil 2 wird das Prinzip des Minimax eingeführt. Zuerst wird die Induktion angewendet, um den kürzesten Weg durch einen Baum und durch ein Labyrinth zu finden. Die Suche für den kürzesten Weg wird dann in einem einfachen Spiel umgewandelt, um das Prinzip des Minimax zu entwickeln.

2.3 Begründung der gewählten Sequenz

Rekursion ist eine natürliche Weise, Induktion in einem Programm umzuwandeln, insbesondere in Fällen, in denen die Induktion zu einem exponentiellen Wachstum der Operationen führt. Rekursion ist jedoch nicht notwendig. Wir entwickeln hier einen alternativen Weg, um Induktion in einem Programm zu implementieren, der keine Rekursionskenntnisse erfordert.

Teil I
Türme von Hanoi ohne Rekursion

Die “Türme von Hanoi” ist ein Spiel, das allein gespielt wird und sehr einfach zu lernen ist. Wenn man das induktive Denken versteht, kann man es sofort lösen und es kann

auch schnell von sehr jungen Kindern gelernt werden. Ohne Induktion ist es aber sehr kompliziert zu lösen.

Es gibt kommerzielle Spiele der Türme von Hanoi, aber man kann es auch sehr gut mit 8 mit den Zahlen 1 bis 8 numerierte Stück Papiere spielen, die die SuS selbst basteln.

3 Verlauf Teil 1

Zuerst müssen die SuS "Die Türme von Hanoi" spielen, bis sie es mit 4 Karten spielen können. Die schnelle SuS können probieren, es bis mit 8 Karten zu spielen. Sie müssen dann die Lösung induktiv ausdrücken.

Sie berechnen mit der Induktion die Anzahl Bewegungen der Karten.

Schlussendlich schreiben Sie ein Programm, um die Turme von Hanoi mit einer Schleife und ohne Rekursion zu programmieren. Anstatt der Rekursion speichert das Programm in eine Liste, was zu tun ist . Das Programm führt die gespeicherte Aktionen so lange aus, bis die Liste leer ist.

Alle rekursiven Algorithmen können eigentlich auf diese Weise ohne Rekursion programmiert werden: Die Liste ersetzt den Stapel der Aufrufe der Funktion, die bei der Rekursion vom Compiler verwaltet wird. Wir werden ein anderes Beispiel dieser Strategie bei der Behandlung des Minimax sehen.

Funktionen, die die Lösung mit Turtle animieren, sind gegeben, um das Verfahren ersichtlich zu machen, damit die SuS visualisieren können, ob ihre Lösung stimmt.

Teil II
Minimax

4 Verlauf Teil 2

4.1 Der Baum

Die SuS suchen eine induktive Antwort zur Frage: "Wie lang ist der kürzeste Weg durch einen Baum von der Wurzel zu einem Blatt?".

Die Antwort lautet: $L(p) = \min_{q \in \mathcal{N}(p)} L(q) + 1$

wobei $\mathcal{N}(p)$ die Menge der Knoten ist, die direkt mit p verbunden sind.

Die SuS lernen den Baum mit Listen von Kindknoten darzustellen. Sie entwickeln ein induktives Verfahren, um Pfade mit wachsenden Längen zu erzeugen, bis sie einen Pfad zu einem Blatt finden. Sie schreiben dann ein Programm, das auf diesem Verfahren basiert ist, um den kürzersten Weg durch den Baum zu bestimmen.

Bemerkung: Die Implementierung des Induktionsverfahren, die in der Aufgabe vorgeschlagen wird, ist nicht die einfachste, um den kürzesten Pfad durch einen Baum zu finden. Die ausgewählte Vorgehensweise ist aber induktiv und kann zur nächsten Aufgaben mit Labyrinthen erweitert werden.

4.2 Das Labyrinth

Als nächstes wird den SuS ein Labyrinth gezeigt. In diesem Labyrinth herausführen mehrere Wege zum Ziel, aber mit unterschiedlichen Streckenlängen. Die SuS müssen finden, inwiefern diese Aufgabe der Baumaufgabe ähnelt und sie sich davon unterscheidet. Sie müssen auch eine induktive Antwort zur Frage: "Wie lang ist der kürzeste Weg durch das Labyrinth?" finden.

Die Antwort lautet: $L(p) = \min_{q \in \mathcal{N}(p)} L(q) + 1$
wobei $\mathcal{N}(p)$ die Menge der Knoten ist, die direkt mit p verbunden sind, aber die noch nicht besucht wurden.

Die SuS erweitern das induktive Verfahren, das für Bäume entwickelt wurde, zu Labyrinthen und passen das Programm zum Baum an das Labyrinth an, um den kürzesten Weg durch das Labyrinth zu finden.

4.3 Ein einfaches Spiel

Nächst müssen die SuS das folgende Spiel mit 2 Spielern betrachten: Spieler 2 darf die erste Strecke durch einen Baum auswählen, Spieler 1 wählt den Rest des Weges aus. Spieler 1 hat das Ziel, den kürzesten Weg zu erhalten. Spieler 2 muss das behindern und so spielen, damit Spieler 1 einen möglichst langen Weg hat. Die SuS müssen die vorherige induktive Formel des Baumes anpassen, um die Länge des resultierenden Weges durch dem Baum auszudrücken.

Die Antwort lautet: $L = \max_{p \in \mathcal{N}(r)} L(p) + 1$
wobei r die Wurzel ist und $\mathcal{N}(r)$ die Menge der Kindknoten. $L(p)$ ist die Länge des kürzesten Pfades und ist wie oben definiert.

4.4 Erweiterung des Spieles

Das Spiel des Baumes wird wie folgt erweitert: Spieler 1 und 2 ziehen abwechselnd dieselbe Spielfigur durch den Baum. Jeder Spieler zieht die Spielfigur um einen Knoten weiter. Spieler 1 versucht, die geringstmögliche Anzahl an Zügen bis zu einem Blatt zu erreichen. Spieler 2 muss so spielen, dass er Spieler 1 behindert, d. h. den Weg verlängert.

Die SuS müssen einen induktiven Ausdruck finden, um die Anzahl Züge bis zum Ziel zu berechnen.

Die Antwort lautet:
Wenn Spieler 1 dran ist: $L_1(p) = \min_{q \in \mathcal{N}(p)} \max_{r \in \mathcal{N}(q)} L_1(r) + 2$
Wenn Spieler 2 dran ist: $L_2(p) = \max_{q \in \mathcal{N}(p)} \min_{r \in \mathcal{N}(q)} L_2(r) + 2$

Diese Ausdrücke führen zu einem Programm, um die besten Züge beider Spieler zu bestimmen. Ähnlich zum Programm der Türme von Hanoi verwendet das Programm eine Liste von Aktionen, die zu erledigen sind. Aktionen, die nicht sofort erledigen werden können, werden mit einfacheren Aktionen ergänzt. Die Aktionen werden durchgeführt, bis die Liste von Aktionen leer ist.

Bermerkung: Als alternativen Algorithmus könnte man die Tiefe von jedem Knoten finden und die Aufgabe vom tiefsten Knoten her lösen und den Baum erklimmen, anstatt von der Wurzel her anzufangen. Die gewählte Implementierung bietet aber eine Verstärkung der behandelten Konzepte.

5 Informationen zur Online Plattform

Die Programmieraufgaben sind am besten auf der Online Plattform geführt. Die Plattform enthält zusätzliche Anweisungen und Hinweise. Die Funktionen der Projekte können unabhängig voneinander geschrieben und getestet werden. Wenn die Zeit in der Klasse begrenzt ist, können die SuS auch nur ein Teil der Funktionen in der Plattform programmieren und die Funktionen trotzdem durchführen und das ganze Programm testen.

Die Plattform kann auch die Arbeit der SuS überprüfen und enthält Testdaten.

Eine Demo der Plattform mit den 3 Projekten zur Infuktion ist auf der folgenden Seite zugänglich:
URL: https://www.uebeplan.ch/mo/inftrain?class=411618358
Benutzername: demo
Passwort: 50369

Die Lösungen der Projekte können direkt auf dem Plattform angeschaut und ausgeführt werden.

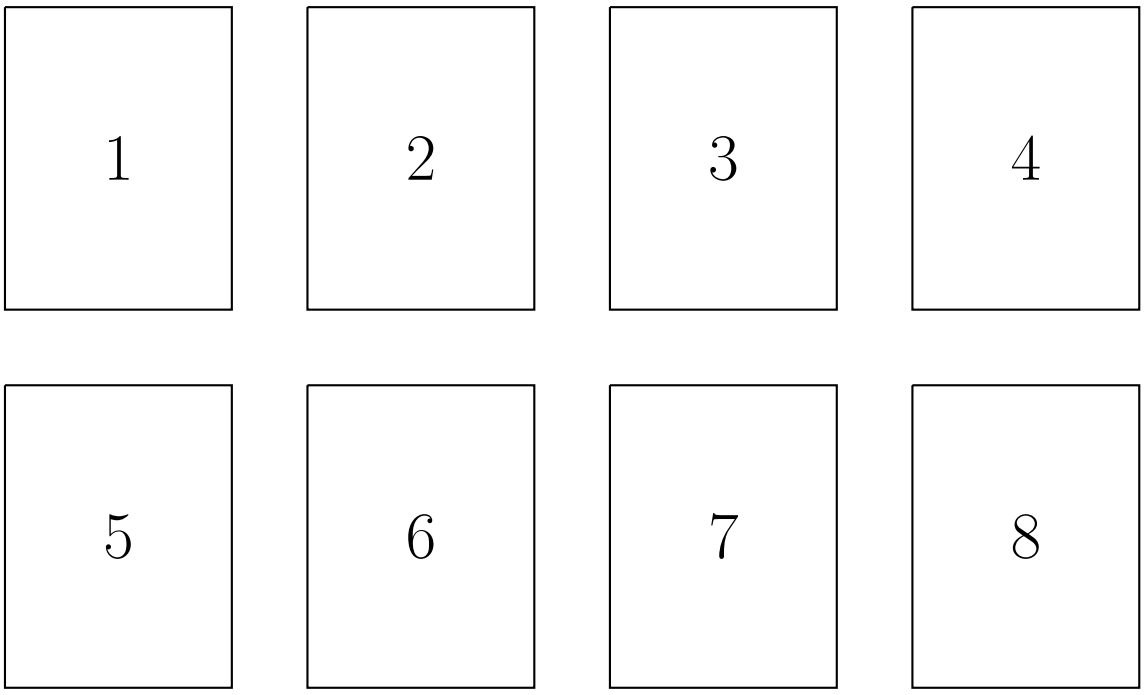
Um die graphische Animationen (Projekt Türme von Hanoi) zu sehen, muss man ganz nach unten das Bildschrim verschieben.

Bermerkung: Die Online Platform unterstützt im Moment das Browser Microsoft Edge nicht. Sie ist am besten mit Chrome, Safari oder Firefox benutzt.

Aufgaben zum Teil 1: Türme von Hanoi

Aufgabe 1.1: Türme von Hanoi spielen

Schneide die folgenden Karten aus und spiele die "Türmen von Hanoi", zuerst mit 3 Karten, dann mit 4 Karten. Wenn du Zeit und Lust dazu hast, kannst du die Anzahl Karten erhöhen, bis du das Spiel mit allen 8 Karten schaffst.



Regeln des Spieles

Wir haben drei Stapel neben einander. Am Anfang haben wir alle Karten auf dem ersten Stapel und die Stapel 2 und 3 sind leer. Die Karten sind in der Reihenfolge der Zahlen geordnet, mit der Karte mit der grössten Zahl ganz unten und die Karte 1 ganz oben. Das Ziel des Spiels ist, alle Karten auf den 3. Stapel zu bringen, wobei wir nur jeweils eine Karte bewegen dürfen und wir keine Karte auf einem Stapel mit einer kleineren Zahl stellen dürfen.

Hier ist eine Illustration des Spieles mit 8 Karten:

Aufgabe 1.2: Berechnungen mit Induktion

1. Erkläre, wie du das Spiel mit Induktion lösen kannst.
   Hinweis: Wenn du das Spiel mit n-1 Karten schaffst, erkläre wie du es mit n Karten auch schaffen könntest.

2. Sei B(n) die Anzahl Bewegungen von Karten, um einen Stapel mit n Karten von einer Stelle zu einer anderen zu bewegen. Bestimme B(8) mit Induktion.
   Hinweis: Wenn du wissen würdest, wie viel B(7) beträgt, könntest du B(8) berechnen?

3. Kannst du eine allgemeine induktive Formel für B(n) schreiben?

4. Kannst du die Formel von c) auch explizit darstellen?
   Hinweis: stelle die Anzahle B(1), B(2), B(3), ... in Binärform dar. Siehst du, wie die induktive Formel die Binärdarstellung verändert?

Aufgabe 1.3: Induktives Verfahren

Erstelle eine Liste von allen Karten Bewegungen, die du machen musst, um 4 Karten von Stapel 1 zum Stapel 3 zu bewegen. Um diese Liste zu schreiben, mach das Folgende:

1. Schreibe zuerst in einem Textdokument, was du zu tun hast:
   Karten 1 bis 4 von Stapel 1 nach Stapel 3 verschieben

2. Lösche diese Zeile und ersetze sie mit dem Induktionsschritt:
   Karten 1 bis 3 von Stapel 1 nach Stapel 2 verschieben
   Karten 4 von Stapel 1 nach Stapel 3 verschieben
   Karten 1 bis 3 von Stapel 2 nach Stapel 3 verschieben

3. Mach weiter, solange der Induktionsschritt ausgeführt werden kann.

4. Du stoppst, wenn alle Zeilen der Bewegung einer Karte entsprechen.

Überprüfe am Ende, dass du $2^4 - 1 = 15$ Verschiebungen hast.

Aufgabe 1.4: Kodierung der auszuführenden Tätigkeiten

Wir möchten ein Programm schreiben, um die "Türme von Hanoi" mit 4 Karten zu implementieren. Dafür müssen wir zuerst speichern, was zu tun ist. Wir verwenden die folgende Kodierung: wir speichern in einer Liste die erste zu bewegende Karte a , die letzte zu bewegende Karte b, den Stapel i, wohin die Karten sich befinden und den Zielstapel j, wo die Karten bewegt werden müssen.

Zum Beispiel bedeutet $[1, 4, 1, 3]$, dass das Programm die Karten 1-4 vom Stapel 1 zum Stapel 3 bewegen muss: das ist eigentlich das Ziel des Programms und wird die erste auszuführende Tätigkeit sein.
$[1, 3, 2, 1]$ bedeutet, dass die Karten 1 bis 3 vom Stapel 2 zum Stapel 1 bewegt werden müssen.

Die Tätigkeiten, die durchgeführt werden müssen, werden in einer Liste *actions* gespeichert, damit die erste zu erledigende Tätigkeit am Ende der Liste steht.

Wenn das Programm eine Tätigkeit am Ende der Liste findet, die mehrere Karten einbezieht, wird es die Tätigkeit mit der folgenden induktiven Formel ersetzen: Die Tätigkeit $[a, b, i, j]$, um die Karten a bis b vom Stapel i zum Stapel j zu bewegen, wird ersetzt mit:

1. $[a, b-1, i, k]$ um die Karten a bis $b-1$ vom Stapel i zum Stapel k zu bewegen (k ist der übrig bleibende Stapel)
2. $[b, b, i, k]$ um die Karte b vom Stapel i zum Stapel j zu bewegen
3. $[a, b-1, k, j]$ um die Karten a bis $b-1$ vom Stapel k zum Stapel j zu bewegen

Zum Beispiel wird die Tätigkeit $[1, 3, 2, 3]$ mit den Tätigkeiten $[1, 2, 2, 1]$, $[3, 3, 2, 3]$, $[1, 2, 1, 3]$ ersetzt.

Zudem werden die Tätigkeiten in umgekehrten Reihenfolge am Ende der Liste der Tätigkeiten *actions* gespeichert.

Wenn das Programm eine Tätigkeit der Art $[a, a, i, j]$ am Ende der Liste *actions* findet, kann es direkt die Karte a vom Stapel i zum Stapel j bewegen und die Tätigkeit löschen.

a) Simuliere das Program. Fange mit der Liste actions = $[[1, 4, 1, 3]]$ an. Wiederhole:

- Wenn die letzte Tätigkeit der Liste *actions* nicht die Form $[a, a, i, j]$ hat, ersetze diese Tätigkeit mit dem Induktionsschritt, wie du es in der Aufgabe 1.3 gemacht hast, aber mit der Kodierung in Listen. Schreibe die 3 Tätigkeiten des Induktionsschrittes in umgekehrten Reihenfolge, damit die erste zu führende Tätigkeit am Ende der Liste steht. Zum Beispiel musst du die Karte $[1, 4, 1, 3]$ mit den Karten $[1, 3, 2, 3], [4, 4, 1, 3], [1, 3, 1, 2]$ ersetzen.
- Wenn die letzte Tätigkeit der Liste von Tätigkeiten die Form $[a, a, i, j]$ hat, mache die entsprechende Bewegung mit der Karte des Spieles und entferne die Tätigkeit der Liste von Tätigkeiten.

Überprüfe, dass, wenn die Liste von Tätigkeiten leer ist, die 4 Karten vom Stapel 1 zum Stapel 3 bewegt wurden.

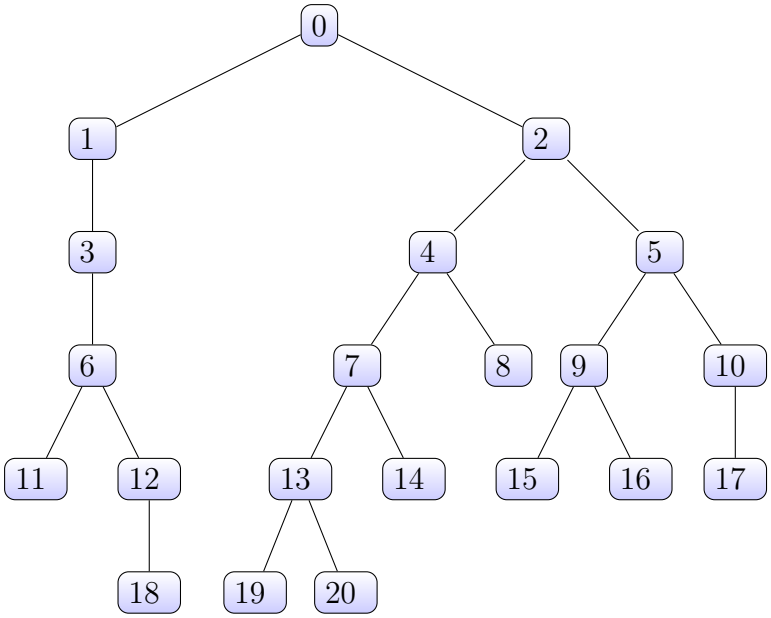
b) Erkläre, warum es effizienter ist, die Tätigkeiten in umgekehrter Reihenfolge zu speichern, damit die Tätigkeit, die zuerst erledigt werden muss, am Ende der Liste steht.

Aufgabe 1.5: Projekt "Türme von Hanoi"

Du findest die detaillierten Anweisungen zum Programm auf der Online-Platform.

Aufgaben zum Teil 2: Baum und Labyrinth

Aufgabe 2.1: Kürzester Pfad durch einen Baum

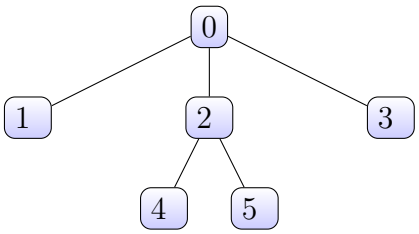


a) Zeichne alle möglichen Pfade durch den Baum von der Wurzel zu einem Blatt. Wie viele Pfade gibt es?
b) Schreibe neben jedem Knoten die Anzahl Pfade von diesem Knoten aus zu einem der Blätter des Baumes.
c) Schreibe einen induktiven Ausdruck, um die Anzahl Pfade durch den Baum zu bestimmen. Hinweis: Drücke die Anzahl Pfade aus einem Knoten p des Baumes $A(p)$ mit der Anzahl Pfade aus den Kindknoten $q \in \mathcal{N}(p)$ aus, wobei $\mathcal{N}(p)$ die Menge der Kindknoten von p ist.
d) Finde den kürzesten Pfad von Knoten 0 (Wurzel) zu einem der Blätter des Baumes.
e) Schreibe neben jedem Knoten die Länge des kürzesten Pfades von diesem Knoten aus zu einem Blatt des Baumes. Zum Beispiel, der Pfad 0-1-3-6-11 hat eine Länge von 4 (= Anzahl Kanten).
f) Finde einen Ausdruck, um die Länge des kürzersten Pfades induktiv auszudrücken. Hinweis: Drücke die Länge $L(p)$ des kürzersten Pfades von einem Knoten p des Baumes mit den Längen der kürzesten Pfade von den Knoten $q \in \mathcal{N}(p)$, wobei $\mathcal{N}(p)$ die Menge der Kindknoten ist.

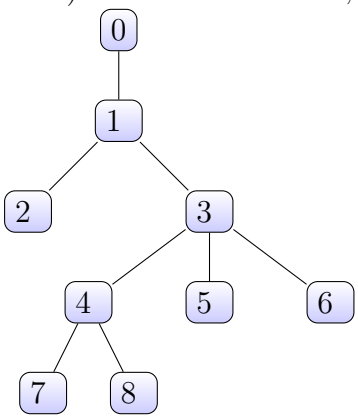
Aufgabe 2.2: Darstellung eines Baumes mit einer Liste

Wir möchten ein Programm schreiben, um in einem Baum den kürzesten Pfad zu einem Blatt in einem Baum zu finden, und wir müssen dafür den Baum im Programm speichern. Der Baum wird in einer Liste von Listen gespeichert. Die Elemente der Hauptliste entsprechen den Knoten des Baumes. Die Knoten werden von 0 her nummeriert. Jedes Element dieser Liste enthält eine Liste der Kindknoten des entsprechenden Knotens. Knoten 0 wird immer die Wurzel des Baumes sein. Die Elemente der Haupliste, die den Blättern

des Baumes entsprechen, werden leere Listen sein.
Zum Beispiel entspricht baum = $[[1, 2, 3], [], [4, 5], [], [], []]$ dem folgenden Baum:



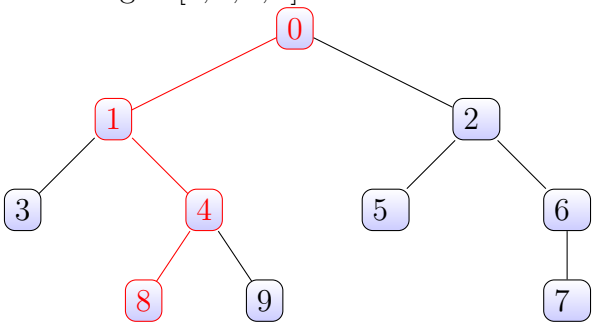
a) Schreibe eine Liste, die dem folgenden Baum entspricht:



b) Zeichne den Baum, der der folgenden Liste entspricht:
baum = $[[1, 2], [3], [4, 5, 6], [7, 8], [], [], [9], [], [], []]$

Aufgabe 2.3: Darstellung eines Pfades des Baumes

Um den kürzesten Pfad zu finden, müssen wir die möglichen Pfade durch den Baum speichern. Die Pfade werden als Listen von Knoten gespeichert. Zum Beispiel entspricht die Liste weg = $[0, 1, 4, 8]$ dem rot markierten Pfad im folgenden Baum:



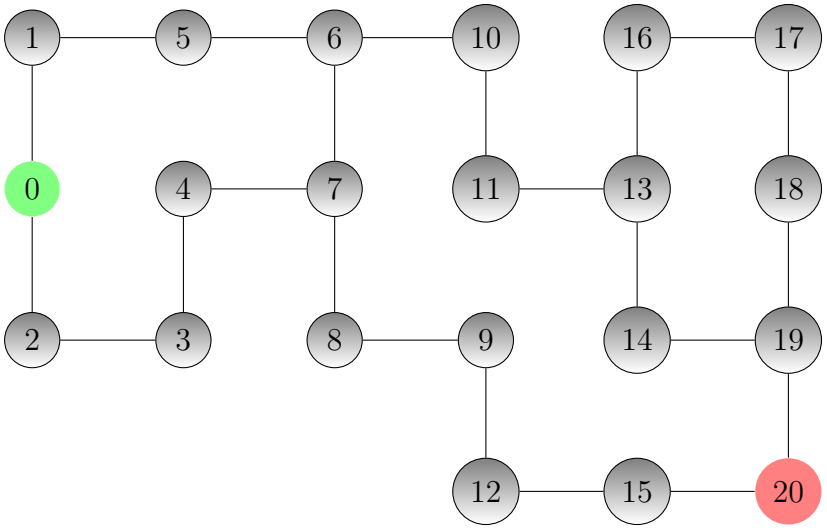
Die möglichen Pfade können in einer Liste gespeichert werden.
a) Schreibe eine Liste mit allen möglichen Pfaden durch den oberen Baum.
b) Unterstreiche die Unterlisten, die den kürzesten Pfaden entsprechen.
c) Sei $N(i)$ die Menge aller Pfade von der Wurzel her von Länge i (die Pfade müssen nicht bis zu einem Blatt gehen). Beschreibe ein induktives Verfahren, um die Menge aller Pfade von der Wurzel her von Länge $i + 1$ zu generieren.

Aufgabe 2.4: Projekt Baum

In diesem Projekt schreiben wir ein Programm, um den kürzesten Pfad zu einem Blatt zu suchen. Wir werden mit dem Pfad von Länge 0 anfangen, der nur die Wurzel enthält und werden mit einem induktiven Verfahren alle Pfade von Länge 1, 2, 3 usw generieren, bis wir einen Pfad zu einem Blatt finden.

Jetzt kannst du auf der Online-Plattform das Projekt Baum programmieren.

Aufgabe 2.5: Kürzester Pfad durch ein Labyrinth

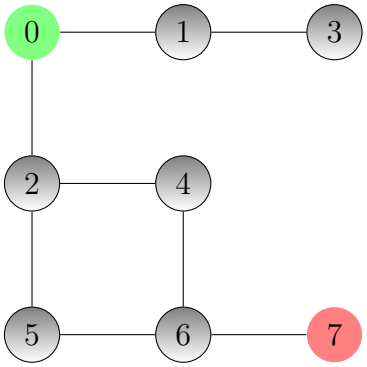


a) Finde den kürzersten Pfad von 0 bis 20. Wie lang ist dieser Pfad?
b) Schreibe neben jedem Knoten die Länge des kürzesten Pfades (= Anzahl Strecken) vom Anfangspunkt zu diesem Knoten.
c) Finde einen Ausdruck, um die Länge des kürzesten Pfades induktiv auszudrücken.
Hinweis: Drücke die Länge $L(p)$ des kürzesten Pfades zu einem Knoten p des Labyrinths mit den Längen den kürzesten Pfaden von den Knoten $q \in \mathcal{N}(p)$, wo $\mathcal{N}(p)$ die Menge der Knoten, die mit p verbunden sind, ist.
d) Vergleiche diese Aufgabe mit der Aufgabe des Baumes. Inwiefern ähneln sie sich und inwiefern unterscheiden sie sich?

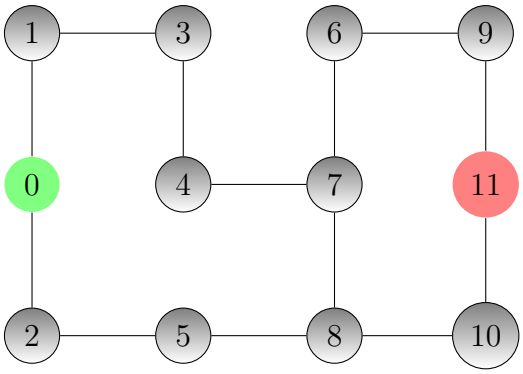
Aufgabe 2.6: Darstellung eines Labyrinthes mit einer Liste

Die obere Darstellung eines Baumes kann auch verwendet werden, um ein Labyrinth darzustellen. In einem Labyrinth ist es aber nicht möglich, untergeordnete Knoten (= Kindknoten) und übergeordneten Knoten zu unterscheiden, da es keine bestimmte Richtung durch das Labyrinth gibt. Die Unterlisten müssen deshalb alle verbundene Knoten enthalten. Anstatt von einer Wurzel und von Blättern gibt es einen Startknoten und einen Zielknoten. Der Startknoten wird immer Knoten 0 entsprechen. Der Zielknoten wird immer dem letzten aufgelisteten Knoten entsprechen.

Beispiel: Der folgende Baum kann wie folgt dargestellt werden:
$[[1, 2], [3], [4, 5], [1], [2, 6], [2, 6], [4, 5, 7], [7]]$



a) Schreibe die Liste, die dem folgenden Labyrinth entpricht:



b) Zeichne ein Labyrinth, das der folgenden Liste entspricht. Bemerkung: Das Aussehen des Labyrinths kann sehr unterschiedlich sein. Das Wichtige ist, dass die Verbindungen der Knoten stimmen.
labyrinth = $[[1, 3], [0, 2, 5], [1, 6], [0, 4, 5], [], [1, 3, 6, 7], [2, 5, 8, 10], [5, 8], [6, 7, 9], [8, 10], [6, 9]]$

Aufgabe 2.7: Projekt Labyrinth

Ähnlich zum Projekt Baum schreiben wir hier ein Programm, um den kürzesten Pfad vom Anfangspunkt zum Zielpunkt zu bestimmen. Wir werden mit dem Pfad von Länge 0 anfangen, der nur den Anfangspunkt enthält, und mit einem induktiven Verfahren alle Pfade von Länge 1, 2, 3 usw generieren, bis wir einen Pfad zum Zielpunkt finden.

a) Welche Veränderungen musst du im Projekt Baum unternehmen, um den kürzesten Pfad in einem Labyrinth zu finden?

b) Es ist unnötig, einen schon besuchten Knoten nochmals zu besuchen, da es nur zu einer Verlängerung des Pfades führt. Wie kannst du überprüfen, ob ein Knoten schon besucht wurde?

c) Jetzt kannst du auf der Online-Plattform das Projekt Labyrinth programmieren.

Aufgabe 2.8: Ein einfaches Spiel

Das folgende Spiel verwendet den Baum von Aufgabe 2.5 und eine Spielfigur und wird von 2 Spielern gespielt. Die 2 Spieler ziehen die gleiche Figur durch den Baum. Das Ziel

von Spieler 1 ist in der geringstmöglichen Anzahl Züge ein Blatt zu erreichen. Das Ziel von Spieler 2 ist Spieler 1 zu behindern, d.h. den Pfad möglichst zu verlängern.

In der einfachsten Version des Spieles darf Spieler 2 anfangen und die Spielfigur um einen Knoten bewegen. Spieler 1 kann dann die Spielfigur bis zum Ziel bewegen.

a) Bestimme den resultierenden Pfad, wenn beide Spieler so gut wie möglich spielen. Benutze dafür die Länge der Pfade, die du in Aufgabe 2.1 e) bestimmt hast.

b) Erkläre, was die Strategie von Spieler 2 sein sollte.

c) Finde einen Ausdruck, um die Länge des resultierenden Pfades zu bestimmen. Du kannst den Ausdruck auf dem induktiven Ausdruck von Aufgabe 2.1 f) basieren.

Aufgabe 2.9: Erweitertes Spiel

In einer erweiterten Version des Spieles spielen die Spieler abwechslend und bewegen jeweils die Spielfigur um einen Knoten nach unten durch den Baum: Zuerst spielt Spieler 1, dann Spieler 2, dann Spieler 1 usw.

a) Bestimme den resultierenden Pfad, wenn beide Spieler so gut wie möglich spielen.

b) Kreise in grün die Knoten ein, bei welchen Spieler 1 dran ist. Kreise in rot die Knoten ein, bei welchen Spieler 2 dran ist. Fange mit den Blättern an und erklimme den Baum. Schreibe jeweils für jeden Knoten die Länge des resultierenden Pfades von diesem Knoten her, wenn beide Spieler so gut wie möglich spielen. Überprüfe, dass deine Lösung mit deinem Resultat in a) übereinstimmt.

c) Schreibe einen induktiven Ausdruck, um die Länge des resultierenden Pfades von einem beliebigen Knoten zu bestimmen, wenn Spieler 1 dran ist.

d) Schreibe einen induktiven Ausdruck, um die Länge des resultierenden Pfades von einem beliebigen Knoten zu bestimmen, wenn Spieler 2 dran ist.

Aufgabe 2.10: Darstellung der Aktionen des Spiels des Baumes

Ähnlich zum Programm der Türme von Hanoi werden wir eine Liste von Aktionen erstellen, die durchgeführt werden müssen, um das induktive Verfahren umzusetzen. Wenn das Programm einer Aktion begegnet, die es nicht sofort erledigen kann, wird es zusätzliche Aktionen am Ende der Liste hinzufügen, die zuerst durchgeführt werden müssen, damit die ursprüngliche Aktion ausgeführt werden kann.

Es gibt zwei Arte von Aktionen:

$[i,$" max"$]$ bedeutet: Berechne die Länge des Pfades bis zum Knoten i, indem man den Pfad durch den Kindknoten mit dem längsten resultierenden Pfad auswählt.

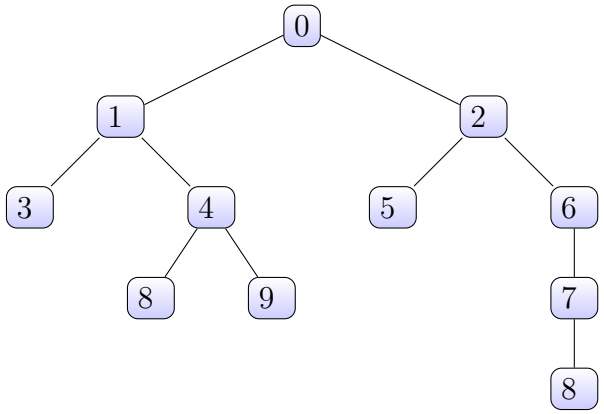
$[i,$" min"$]$ bedeutet: Berechne die Länge des Pfades bis zum Knoten i, indem man den Pfad durch den Kindknoten mit dem kürzesten resultierenden Pfad auswählt.

Die Aktionen werden immer vom Ende der Liste her ausgeführt.

Aktionen $[i,$" max"$]$ und $[i,$" min"$]$ können erst durchgeführt werden, wenn die Länge des Pfades bis zu allen Kindknoten von Knoten i schon bestimmt wurde. Wenn es nicht der Fall ist, kann die jeweilige Aktion nicht erledigt werden. In diesem Fall muss das Programm die Liste von Aktionen ergänzen, um die Länge der Pfade zu den Kindknoten zuerst zu bestimmen. Wenn die jeweilige Aktion der Art *max* ist, müssen die Aktionen der Kindknoten der Art *min* sein und umgekehrt.

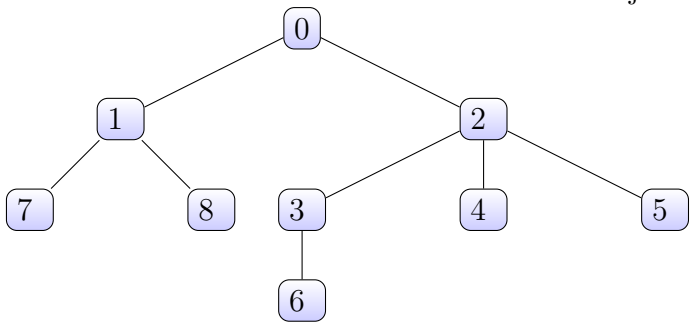
Beispiel: Am Anfang des Programms besteht die Liste von Aktionen aus einer einzelnen Aktion: actions = $[[0,$" min"$]]$

Da die Längen der Pfade zu den Kindknoten 2 und 3 noch nicht bekannt sind, müssen zuerst die Aktionen [2," max"] und [3," max"] durchgeführt werden. Die Aktionsliste wird entsprechend ergänzt:
$actions = [[0,"\ \min"], [2,"\ \max"], [3,"\ \max"]]$
Die Aktion [3," max"] kann noch nicht erledigt werden, da wir die Längen der Pfade zu den Kindknoten 5 und 6 noch nicht kennen. Deshalb müssen wir die Aktionsliste wie folgt ergänzen:
$actions = [[0,"\ \min"], [2,"\ \max"], [3,"\ \max"], [5,"\ \min"], [6,"\ \min"]]$
Diese Vorgehensweise wird weitergeführt bis einem Blatt des Baumes begegnet wird. Da ein Blatt kein Kind hat, kann die Länge des Pfades sofort bestimmt werden (0) und die entsprechende Aktion wird von der Liste der Aktionen entfernt.
Simuliere die Durchführung der Aktionen mit dem unteren Baum. Schreibe jeweils die Länge der Pfade auf dem Baum, wenn sie vom Algorithmus berechnet werden:



Aufgabe 2.11: Projekt Spiel des Baumes

Nun kannst du auf der Online-Plattform das Projekt Spiel des Baumes programmieren.



Lösung Aufgabe 1

Aufgabe 1.2

1. Die induktive Lösung des Spieles lautet:

   Wir möchten die Karten 1-8 vom Stapel 1 zum Stapel 3 umstellen.

   Wir wissen nicht, wie man Karten 1-8 umstellt, aber wenn wir wissen würden, wie man Karten 1-7 umstellt, könnten wir die Karten 1-7 auf Stapel 2 bringen, Karte 8 auf Stapel 3 stellen und die Karten 1-7 auf Stapel 3 auf Karte 8 umstellen. Auf diese Weise könnte man Karten 1-8 auf Stapel 3 umstellen.

   Wir wissen nicht, wie man Karten 1-7 auf einen anderen Stapel umstellt, aber wenn wir wissen würden, wie man die Karten 1-6 umstellen könnte, dann könnten wir sie auf den übrigbleibenden Stapel umstellen, dann Karte 7 auf den Zielstapel legen und die Karten 1-6 auf Karte 7 umstellen.

   Wir können diese Überlegung fortsetzen und weiter das Problem reduzieren, bis zum Problem, bei dem wir nur wissen müssen, wie wir eine einzige Karte auf einen anderen Stapel verschieben können, was selbstverständlich ist.

   Im Allgemeinen, würden wir wissen, wie man einen Stapel mit n-1 Karte umstellt, dann könnte man einen Stapel mit n Karten umstellen, indem man n-1 Karte auf den übrigbleibenden Stapel bringt, die n-te Karte auf den Zielstapel stellt, und den Stapel mit n-1 Karte darauf umstellt.

   Um eine Karte umzustellen, kann man sie einfach bewegen, deshalb wissen wir durch vollständige Induktion wie man n Karten umstellt.

   Um 8 Karten zu bewegen, muss man 7 Karten auf den übrigbleibenden Stapel umstellen, die 8. Karten auf den Zielstapel umstellen, und schliesslich die 7 Karten auf die 8. Karte umstellen. Das heisst:
   $B(8) = B(7) + 1 + B(7) = 2 \cdot B(7) + 1$
   Auf die gleiche Weise haben wir:
   $B(7) = 2 \cdot B(6) + 1$
   $B(6) = 2 \cdot B(5) + 1$
   usw

2. Im Allgemeinen haben wir $B(n) = 2 \cdot B(n-1) + 1$

3. Wir werden hier die Anzahl Bewegungen in Binärform darstellen.
   Wir haben $B(1) = 1_2$ (um eine Karte zu bewegen, brauchen wir nur eine Bewegung!).
   Um $B(2)$ zu berechnen, müssen wir $B(1)$ verdoppeln und nachher 1 addieren. In Binärdarstellung heisst eine Verdopplung, dass man alle Ziffern der Zahl nach links um eine Stelle verschiebt und eine Null auf der rechte Seite schreibt. 1_2 wird zu 10_2. Bei der Addierung von 1, wandeln wir die Null in eine 1, das heisst, $B(2) = 11_2$.
   Um $B(3)$ zu berechnen, müssen wir $B(2)$ verdoppeln, dass heisst die Ziffern nach links verschieben und eine Null hinzufügen. Wenn wir noch 1 addieren, wird die Null zu einer 1, das heisst, $B(3) = 111_2$.
   Wir können dieses Verfahren wiederholen und finden, dass $B(n) = \underbrace{1111...111_2}_{\text{n mal 1}}$.

Wir bemerken noch, dass $\underbrace{1111...111_2}_{\text{n mal 1}} + 1 = 2^n$.

Daraus lässt sich schliessen, dass $B(n) = 2^n - 1$

Aufgabe 1.3

1. *Karte 1 von Stapel 1 nach Stapel 2 verschieben*
2. *Karte 2 von Stapel 1 nach Stapel 3 verschieben*
3. *Karte 1 von Stapel 2 nach Stapel 3 verschieben*
4. *Karte 3 von Stapel 1 nach Stapel 2 verschieben*
5. *Karte 1 von Stapel 3 nach Stapel 1 verschieben*
6. *Karte 2 von Stapel 3 nach Stapel 2 verschieben*
7. *Karte 1 von Stapel 1 nach Stapel 2 verschieben*
8. *Karte 4 von Stapel 1 nach Stapel 3 verschieben*
9. *Karte 1 von Stapel 2 nach Stapel 3 verschieben*
10. *Karte 2 von Stapel 2 nach Stapel 1 verschieben*
11. *Karte 1 von Stapel 3 nach Stapel 1 verschieben*
12. *Karte 3 von Stapel 2 nach Stapel 3 verschieben*
13. *Karte 1 von Stapel 1 nach Stapel 2 verschieben*
14. *Karte 2 von Stapel 1 nach Stapel 3 verschieben*
15. *Karte 1 von Stapel 2 nach Stapel 3 verschieben*

Aufgabe 1.4

a) $[[1, 4, 1, 3]]$
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 3, 1, 2]]$
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 2, 3, 2], [3, 3, 1, 2], [1, 2, 1, 3]]$
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 2, 3, 2], [3, 3, 1, 2], [1, 1, 2, 3], [2, 2, 1, 3], [1, 1, 1, 2]]$
Karte 1 wird von Stapel 1 zu Stapel 2 bewegt
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 2, 3, 2], [3, 3, 1, 2], [1, 1, 2, 3], [2, 2, 1, 3]]$
Karte 2 wird von Stapel 1 zu Stapel 3 bewegt
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 2, 3, 2], [3, 3, 1, 2], [1, 1, 2, 3]]$
Karte 1 wird von Stapel 2 zu Stapel 3 bewegt
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 2, 3, 2], [3, 3, 1, 2]]$
Karte 3 wird von Stapel 1 zu Stapel 2 bewegt
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 2, 3, 2]]$
$[[1, 3, 2, 3], [4, 4, 1, 3], [1, 1, 1, 2], [2, 2, 3, 2], [1, 1, 3, 1]]$
Karte 1 wird von Stapel 3 zu Stapel 1 bewegt

[[1, 3, 2, 3], [4, 4, 1, 3], [1, 1, 1, 2], [2, 2, 3, 2]]
Karte 2 wird von Stapel 3 zu Stapel 2 bewegt
[[1, 3, 2, 3], [4, 4, 1, 3], [1, 1, 1, 2]]
Karte 1 wird von Stapel 1 zu Stapel 2 bewegt
[[1, 3, 2, 3], [4, 4, 1, 3]]
Karte 4 wird von Stapel 1 zu Stapel 3 bewegt
[[1, 3, 2, 3]]
[[1, 2, 1, 3], [3, 3, 2, 3], [1, 2, 2, 1]]
Karte 2 wird von Stapel 2 zu Stapel 1 bewegt
[[1, 2, 1, 3], [3, 3, 2, 3]]
Karte 3 wird von Stapel 2 zu Stapel 3 bewegt
[[1, 2, 1, 3]]
[[1, 1, 2, 3], [2, 2, 1, 3], [1, 1, 1, 2]]
Karte 1 wird von Stapel 1 zu Stapel 2 bewegt
[[1, 1, 2, 3], [2, 2, 1, 3]]
Karte 2 wird von Stapel 1 zu Stapel 3 bewegt
[[1, 1, 2, 3]]
Karte 1 wird von Stapel 2 zu Stapel 3 bewegt
DAS ENDE
b) Es ist effizienter, ein Element am Ende einer Liste wegzunehmen oder hinzuzufügen als am Anfang. Da immer die oberste Karte eines Stapels verändert wird, ist es deshalb effizienter, wenn die oberste Karte am Ende der Liste aufgelistet wird.

Aufgabe 1.5

```
# Türme von Hanoi
import time
import turtle

turtle.restart()
t = turtle.Turtle()
t.speed(10)
t.hideturtle()

_topStack=[0,0,0]
_delta = 6

def drawKarte(a,stack):
    t.width(_delta-1)
    t.penup()
    t.goto(-a*5+(stack-1)*100,_topStack[stack-1]+1)
    t.pendown()
    t.goto(a*5+(stack-1)*100,_topStack[stack-1]+1)
    _topStack[stack-1]+=_delta
    t.penup()

def eraseKarte(a,stack):
```

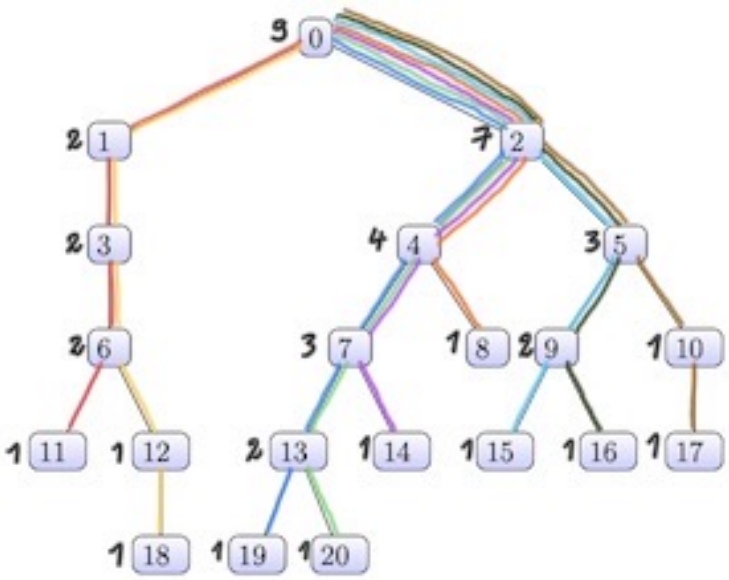
```
    t.width(5)
    t.color("white")
    _topStack[stack-1]-=_delta
    t.penup()
    t.goto(-a*5+(stack-1)*100,_topStack[stack-1]+1)
    t.pendown()
    t.goto(a*5+(stack-1)*100,_topStack[stack-1]+1)
    t.penup()
    t.color("black")

# Erzeugt n Turtles, die die n Karten darstellen
def initHanoi():
    global _topStack
    # Zeichnet die Basen der Stapel
    for i in range(3):
        t.penup()
        t.goto(-40+i*100,-3)
        t.pendown()
        t.goto(40+i*100,-3)
    _topStack=[0,0,0]
    t.hideturtle()
    t.penup()

def drawStack(n):
    for i in range(1,n+1):
        drawKarte(n-i+1,1)

def moveKarte(n,i,j):
    eraseKarte(n,i)
    drawKarte(n,j)

def computeK(i,j):
    return (2*(i-1)+2*(j-1)) % 3 +1

initHanoi()

# CODE FÜR AUFGABE induktion

def induktion(activity):
    a = activity[0]
    b = activity[1]
    i = activity[2]
    j = activity[3]
    k = computeK(i,j)
    if a==b:
        moveKarte(a,i,j)
        return []
```

```
    else:
        return [[a,b-1,k,j],[b,b,i,j],[a, b-1, i, k]]

# ENDE CODE FÜR AUFGABE induktion

# CODE FÜR AUFGABE solve

def solve(n):
    activities = [[1,n,1,3]]
    while len(activities)>0:
        activity = activities.pop();
        activities += induktion(activity)

# ENDE CODE FÜR AUFGABE solve

# CODE FÜR AUFGABE main

drawStack(4)
solve(4)

# ENDE CODE FÜR AUFGABE main

turtle.done()
```

Lösung Aufgabe 2

Aufgabe 2.1

a), b)

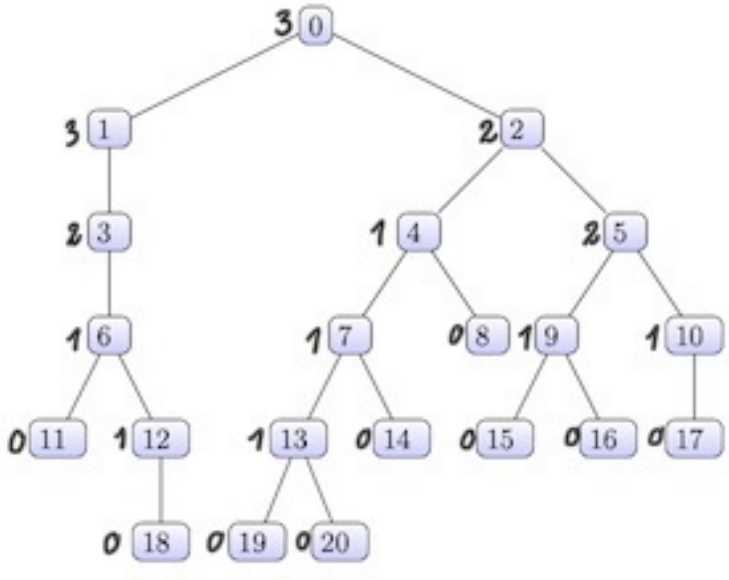


9 Pfade = Anzahl Blätter

c) $A(p) = \sum_{q \in \mathcal{N}(p)} A(q)$

d) $0 - 2 - 3 - 8$

e)

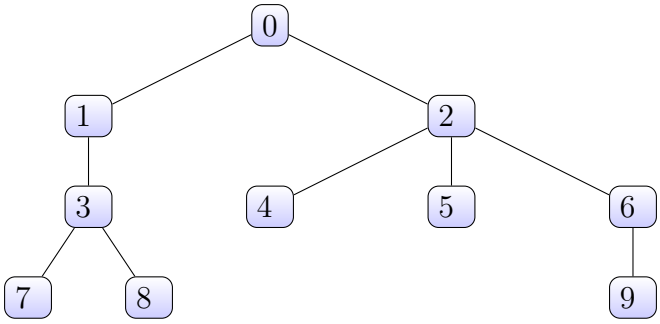


f) $L(p) = \min_{q \in \mathcal{N}} L(q) + 1$

Aufgabe 2.2

a) $[[1], [2, 3], [], [4, 5, 6], [7, 8], [], [], [], []]$

b)



Aufgabe 2.3

a), b) $[[0,1,3],[0,1,4,8],[0,1,4,9],[0,2,5],[0,2,6,7]]$
c) Wir nehmen die Menge aller Pfade mit Länge i. Wir verlängern diese Pfade mit den Kindknoten der letzten Knoten jeder Pfade. Damit erhalten wir alle Pfade mit Länge $i+1$.

Aufgabe 2.4

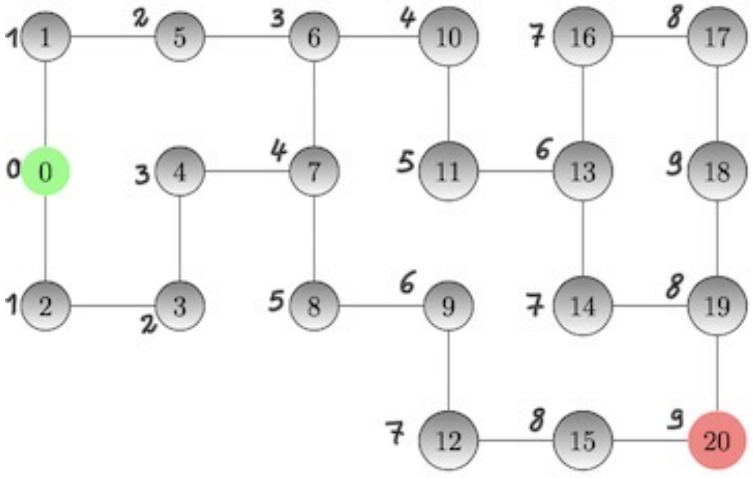
```
tree = [[1,2],[3],[4,5],[6],[7,8],[9,10],[11,12],[13,14],[],[15,16],[17],
```

```
# CODE FÜR AUFGABE growPath

def growPath(tree,path):
    nodes = tree[path[-1]]
    paths = []
    for n in nodes:
        newpath = path.copy()
        newpath.append(n)
        paths.append(newpath)
    return paths

# ENDE CODE FÜR AUFGABE growPath

# CODE FÜR AUFGABE isPathToLeaf

def isPathToLeaf(tree,path):
    if len(tree[path[-1]])==0:
        return True
    else:
        return False

# ENDE CODE FÜR AUFGABE isPathToLeaf

# CODE FÜR AUFGABE findShortestPath

def findShortestPath(tree):
```

```
    paths = [[0]]
    while True:
        # überprüfe, ob die Liste newPaths ein Pfad zu einem Blatt enthält
        for path in paths:
            if isPathToLeaf(tree,path):
                return path
        # erstellen eine neue Liste von Pfaden, um die verlängerten Pfade zu
        newpaths = []
        for path in paths:
            p = growPath(tree,path)
            newpaths += p
        # ersetze die Liste von Pfaden mit der neuen Liste
        paths = newpaths

# ENDE CODE FÜR AUFGABE findShortestPath

# CODE FÜR AUFGABE main

path = findShortestPath(tree)
print(path)
print("Länge des Pfades:",len(path)-1)

# ENDE CODE FÜR AUFGABE main
```

Aufgabe 2.5

a) 0 - 2 - 3 - 4 - 7 - 8 - 9 - 12 - 15 - 20
oder 0 - 1 - 5 - 6 - 7 - 8 - 9 - 12 - 15 - 20
oder 0 - 1 - 5 - 6 - 10 - 11 - 13 - 14 - 19 - 20
Länge: 9
b)

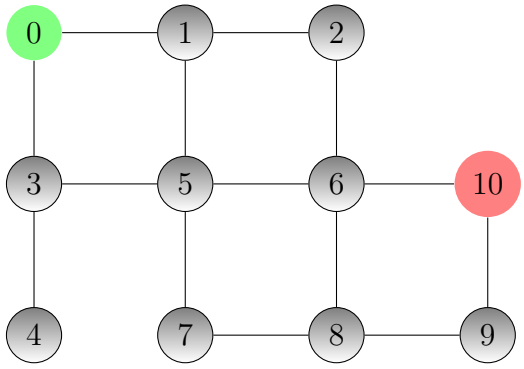


c) $L(p) = \min_{q \in \mathcal{N}(p)} L(q) + 1$
d) Es gibt kein Unterschied zwischen untergeordnete und übergeordnete Knoten. Die Strecken haben keine Richtung. Wir können zum Beispiel gleich gut vom Knoten 7 zum

Knoten 6 gehen, als vom Knoten 6 zum Knoten 7. Anstatt von mehreren Schlusspunkten (Blätter), gibt es nur einen Schlusspunkt (das Ziel).

Aufgabe 2.6

a) $[[1,2],[0,3],[0,5],[1,4],[3,7],[2,8],[7,9],[4,6,8],[5,7,10],[6,11],[8,11],[9,10]]$
b) Zum Beispiel



Aufgabe 2.7

a) Wir müssen überprüfen, ob die Pfade den Zielpunkt enthalten, anstatt von einem Blatt.
b) Wir können überprüfen, ob der neue Knoten schon im Pfad enthalten wird. Wenn es der Fall ist, müssen wir den Pfad durch diesen Knoten nicht zu verlängern.
c)

```
labyrinth = [[1,2],[0,5],[0,3],[2,4],[3,7],[1,6],[5,7,10],[4,6,8],[7,9],[8,1

# CODE FÜR AUFGABE growPath

def growPath(lab,path):
    nodes = lab[path[-1]]
    paths = []
    for n in nodes:
        if n not in path:
            newpath = path.copy()
            newpath.append(n)
            paths.append(newpath)
    return paths

# ENDE CODE FÜR AUFGABE growPath

# CODE FÜR AUFGABE isPathToGoal

def isPathToGoal(lab,path):
    if path[-1]==len(lab)-1:
        return True
```

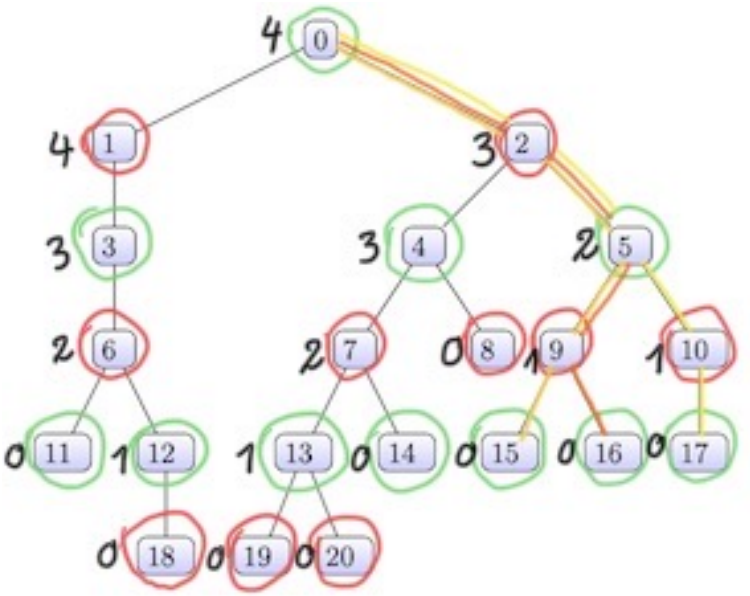
```
    else:
        return False

# ENDE CODE FÜR AUFGABE isPathToGoal

# CODE FÜR AUFGABE findShortestPath

def findShortestPath(lab):
    paths = [[0]]
    while True:
        # überprüfe, ob die Liste newPaths ein Pfad zum Zielpunkt enthält
        for path in paths:
            if isPathToGoal(lab,path):
                return path
        # erstellen eine neue Liste von Pfaden, um die verlängerten Pfade zu
        newpaths = []
        for path in paths:
            p = growPath(lab,path)
            newpaths += p
        # ersetze die Liste von Pfaden mit der neuen Liste
        paths = newpaths

# ENDE CODE FÜR AUFGABE findShortestPath

# CODE FÜR AUFGABE main

path = findShortestPath(labyrinth)
print(path)
print("Länge des Pfades:",len(path)-1)

# ENDE CODE FÜR AUFGABE main
```

Aufgabe 2.8

a) 0 - 1 - 3 - 4 - 6
b) Spieler 2 muss den Kindknoten auswählen, über dem der kürzeste Pfad am längsten ist.
c) $L = \max_{p \in \mathcal{N}(r)} L(p) + 1$

Aufgabe 2.9

a) 0 - 2 - 5 - 9 - 15 oder 0 - 2 - 5 - 9 - 16 oder 0 - 2 - 5 - 10 - 17
b)



c) $L_1(p) = \min_{q \in \mathcal{N}(p)} \max_{s \in \mathcal{N}(q)} L_1(s) + 2$
d) $L_2(p) = \min_{q \in \mathcal{N}(p)} \max_{s \in \mathcal{N}(q)} L_2(s) + 2$

Aufgabe 2.10

```
[[0, "min"]]
[[0, "min"], [1, "max], [2, "max"]]
[[0, "min"], [1, "max], [2, "max"], [5, "min"], [6, "min"]]
[[0, "min"], [1, "max], [2, "max"], [5, "min"], [6, "min"], [7, "max"]]
[[0, "min"], [1, "max], [2, "max"], [5, "min"], [6, "min"], [7, "max"],
[8, "min"]]
Knoten 8: Länge 0
[[0, "min"], [1, "max], [2, "max"], [5, "min"], [6, "min"], [7, "max"]]
Knoten 7: Länge 1
[[0, "min"], [1, "max], [2, "max"], [5, "min"], [6, "min"]]
Knoten 6: Länge 2
[[0, "min"], [1, "max], [2, "max"], [5, "min"]]
Knoten 5: Länge 0
[[0, "min"], [1, "max], [2, "max"]]
Knoten 2: Länge 3
[[0, "min"], [1, "max]]
[[0, "min"], [1, "max], [3, "min"], [4, "min"]]
[[0, "min"], [1, "max], [3, "min"], [4, "min"], [8, "max"], [9, "max"]]
Knoten 9: Länge 0
[[0, "min"], [1, "max], [3, "min"], [4, "min"], [8, "max"]]
Knoten 8: Länge 0
[[0, "min"], [1, "max], [3, "min"], [4, "min"]]
Knoten 4: Länge 1
[[0, "min"], [1, "max], [3, "min"]]
Knoten 3: Länge 0
```

```
[[0, "min"], [1, "max]]
Knoten 1: Länge 2
[[0, "min"]]
Knoten 0: Länge 2
```

Aufgabe 2.11

```
tree = [[1,2],[3],[4,5],[6],[7,8],[9,10],[11,12],[13,14],[],[15,16],[17],

# CODE FÜR AUFGABE haveAllNodesLengths

def haveAllNodesLengths(nodes,lengths):
    for n in nodes:
        if lengths[n]==-1:
            return False
    return True

# ENDE CODE FÜR AUFGABE haveAllNodesLengths

# CODE FÜR AUFGABE findLongestPath

def findLongestPath(nodes,lengths):
    maxLength = -1
    for n in nodes:
        if maxLength==-1 or lengths[n]>maxLength:
            maxLength = lengths[n]
    return maxLength

# ENDE CODE FÜR AUFGABE findLongestPath

# CODE FÜR AUFGABE findShortestPath

def findShortestPath(nodes,lengths):
    minLength = -1
    for n in nodes:
        if minLength==-1 or lengths[n]0:
        action = actions[-1]
        node = action[0]
        actiontype = action[1]
        childnodes = tree[node]
        if not haveAllNodesLengths(childnodes,lengths):
            for n in childnodes:
                actions.append([n, getNextActionType(action[1])])
        else:
            if actiontype=="min":
                length = findShortestPath(childnodes,lengths)+1
```

```
            else:
                length = findLongestPath(childnodes,lengths)+1
            lengths[node] = length
            actions.pop()
    return lengths

# ENDE CODE FÜR AUFGABE findMinimaxLengths

# CODE FÜR AUFGABE extractMinimaxPath

def extractMinimaxPath(tree,lengths):
    path = [0]
    while lengths[path[-1]]!=0:
        lastnode = path[-1]
        childnodes = tree[lastnode]
        for n in childnodes:
            if lengths[n]+1==lengths[lastnode]:
                path.append(n)
                break
    return path

# ENDE CODE FÜR AUFGABE extractMinimaxPath

# CODE FÜR AUFGABE main

lengths = findMinimaxLengths(tree)
minimaxPath = extractMinimaxPath(tree,lengths)
print(minimaxPath)
print(len(minimaxPath)-1)

# ENDE CODE FÜR AUFGABE main
```