Zweite Runde Theorie

Aufgaben

Swiss Olympiad in Informatics

9. März 2019

Anweisungen

- Öffne die Prüfungs erst, wenn du dazu aufgefordert wirst. Die Prüfung beginnt für alle Teilnehmer gleichzeitig und dauert 5 Stunden.
- Mit Ausnahme von Uhren (keine Smartwatches) sind keine elektronischen Geräte auf den Tischen erlaubt. Schalte dein Mobiltelefon aus.
- Beginne jede Aufgabe auf einer neuen Seite und schreibe deinen Namen auf alle Blätter. Nummeriere deine Blätter und sortiere sie vor der Abgabe.
- Verwende keinen Bleistift und schreibe nicht mit rot.
- Schreibe lesbar.
- Schau dir jede Teilaufgabe an, auch wenn du eine vorherige nicht lösen konntest. Einige Teilaufgaben lassen sich auch ohne vorherige Teilaufgabe lösen.

Bewertung

Deine Lösung wird anhand ihrer Korrektheit und der asymptotische Laufzeit und Speichernutzung bewertet, sowie der Begründung dieser Punkte. Wir erwarten, dass du einen Beweis oder eine Beweisskizze für die Korrekteit und Laufzeit/Speichernutzung lieferst.

Falls ein Algorithmus verlangt ist, solltest du dich als Richtlinie an folgende Struktur halten:

1. Beschreibe die Idee hinter deinem Algorithmus so verständlich wie möglich.
2. Erkläre, wieso der Ansatz die Aufgabe korrekt löst.
3. Analysiere die asymptotische Laufzeit und den asymptotischen Speicherverbrauch.
4. Schreibe ein Lösungsprogramm in Pseudocode auf. Du kannst einfache Teile wie Eingabe/Ausgabe weglassen und auch mathematische Ausdrücke benutzen.

Wenn ein Teil deiner Lösung für mehrere Teilaufgaben gilt, reicht es, ihn einmal aufzuschreiben und von da an auf ihn zu verweisen.

Viel Erfolg!

Käse Vorschlag

Maus Stofl macht eine Lehre beim Käse-Grossmeister JK. Leider ist JK sehr geheimnistuerisch, wenn es darum geht, wie man zwei Käse Sorten auswählt, die gut zusammenpassen. Maus Stofl kennt von jedem Käse i seine zwei Eigenschaften a_i und b_i, die Grossmeister JK mit seinem exzellenten Geruchssinn bestimmt hat. Maus Stofl meint jetzt ausserdem, die Formel herausgefunden zu haben, mit der er bestimmen kann, wie gut zwei Käsesorten zusammenpassen. Sie lautet:

$$\frac{\left|a_i - a_j\right| \cdot b_i \cdot b_j}{\max(|b_i|, \left|b_j\right|)} \tag{1}$$

In dieser Formel steht $|x|$ für den Betrag von x (x falls $x \geq 0$ oder $-x$ falls $x < 0$) und $\max(x, y)$ für das Maximum von x und y (x falls $x \geq y$ und y falls $x < y$).

Das Resultat der Formel nennen wir die Empfehlungsnote. Je höher dieses Wert desto besser passen die Käsesorte i und j zusammen. Man kann eine Käsesorte auch mit sich selber kombinieren, in diesem Fall ist der Wert 0 (weil $|a_i - a_i| = 0$). Es kann auch Käsesorten geben, die überhaupt nicht zusammenpassen und eine negative Empfehlungsnote haben.

Maus Stofl möchte Käse Grossmeister JK beeindrucken und möchte, dass du ihm ein Programm schreibst, welches die höchste mögliche Empfehlungnote berechnet.

Formale Beschreibung Gegeben ist die Anzahl Käsesorten n ($2 \leq n$). Für jede Käsesorte sind zwei Ganzzahlen gegeben a_i, b_i ($1 \leq i \leq n$). Finde das Maximum von $\frac{\left|a_i - a_j\right| \cdot b_i \cdot b_j}{\max(|b_i|, \left|b_j\right|)}$.

Teilaufgabe 1: Mach einen Vorschlag (10 Punkte)

Heute gibt es im Laden $n = 9$ Käsesorten. Welche zwei muss Stofl kombinieren, damit die Empfehlungsnote möglichst hoch ist?

i	0	1	2	3	4	5	6	7	8
$\mathbf{a_i}$	1	-1	5	-2	3	5	-6	4	0
$\mathbf{b_i}$	5	1	2	4	7	-3	-2	3	4

Teilaufgabe 2: Zwei Käsesorten auswählen (70 Punkte)

Am nächsten Tag hat es so viele verschiedene Käsesorten, dass Stofl die Berechnung nicht mehr von Hand machen kann. Allerdings fällt ihm auf, dass alle Käsesorten einen positiven b-Wert haben ($b_i \geq 0$).

Entwerfe einen Algorithmus der, gegeben n und sowie die Werte a_i und b_i, die maximale Empfehlungsnote berechnet.

Teilaufgabe 3: Negative Eigenschaften (20 Punkte)

Es können nun auch Käsesorten vorkommen, die negative b-Werte haben.

Angenommen, du kennst eine effiziente Lösung für Teilaufgabe 2, wie müssen wir den Algorithmus modifizieren, damit er für den allgemeinen Fall funktioniert?

Hinweis: Auch wenn du Teilaufgabe 2 nicht lösen konntest, kannst du annehmen das eine effizienter Algorithmus, der das Problem für positive b löst, gegeben ist.

Granatäpfel

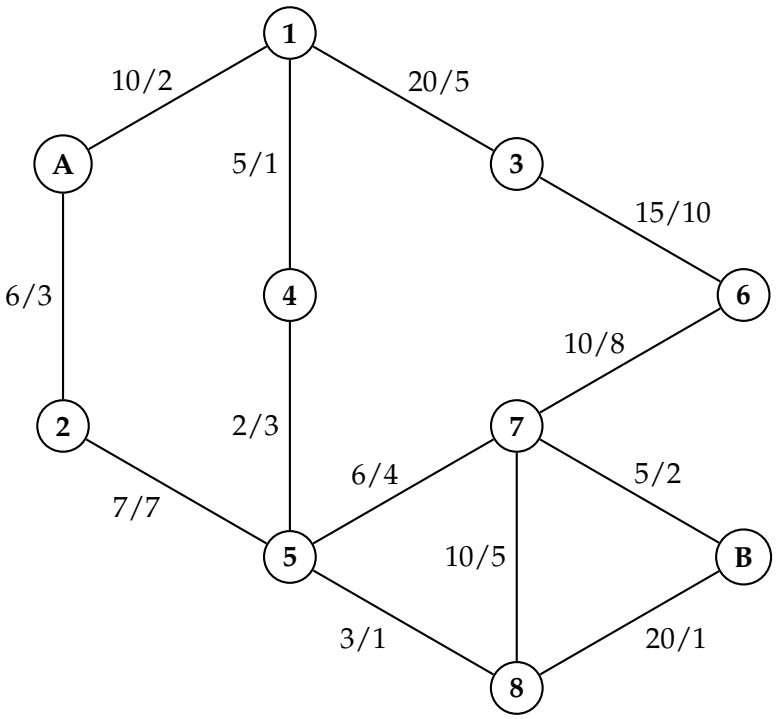
Maus Ada ist Organisator der MOI (Mäuseolympiade in Informatik) in Baku. Einer ihrer Aufgaben ist es, möglichst viele Granatäpfel als Snacks vom Lager zum Wettbewerbsort zu bringen. Um die Granatäpfel innerhalb von Zeit T vom Lager zum Wettbewerbsort zu bringen, benutzt Maus Ada einen Lastwagen, in den beliebig viele Granatäpfel hineinpassen. Das Problem ist, dass jede Strasse in Baku eine Gewichtslimite hat, die der Lastwagen nicht überschreiten darf, um darauf fahren zu dürfen. Da die Granatäpfel natürlich etwas wiegen kann sie nicht beliebig viele mitnehmen . . .

Hilf Maus Ada, einen Weg zu finden, der höchstens T lang ist und bei welchem sie den Laster mit maximal vielen Grantatäpfeln befüllen kann.

Formale Beschreibung Gegeben ist ein Graph, bei dem jede Kante zwei positive ganze Zahlen w und t besitzt. Finde das maximale Gewicht W, sodass es von A nach B einen Pfad gibt, dessen Länge (die Summe der t der Kanten) $\leq T$ ist, und für jede Kante auf den Pfad $w \geq W$ gilt.

Es ist garantiert, dass der Graph zusammenhängend ist und dass immer ein Pfad von A nach B in Zeit bis und mit T existiert.

Teilaufgabe 1: Lösen eines Beispiels (10 Punkte)



Welches ist das maximale Gewicht, sodass der Laster vom Lager aus den Wettbewerbsort in $T = 18$ erreicht und keine Gewichtlimite überschreitet? Das Lager ist mit A

markiert, der Wettbewerbsort mit B. Bei jeder Kante stehen zwei Zahlen, w/t, die linke steht für die Gewichtslimite, und die rechte für die Zeit, die der Laster braucht. Erkläre kurz, wieso es nicht besser geht.

Teilaufgabe 2: Unendlich Zeit (30 Punkte)

Wir nehmen an, dass es egal ist, wann die Granatäpfel am Wettbewerbsort ankommen, also ist $T = \infty$. Die Gewichtslimiten für die Strassen sind positive Ganzzahlen.

Teilaufgabe 3: Zwei Gewichte (20 Punkte)

Das Zeitlimit ist wieder zurück. Jetzt hat jede Strasse jedoch das Gewichtslimit 1 oder 2, das bedeutet $1 \leq w \leq 2$.

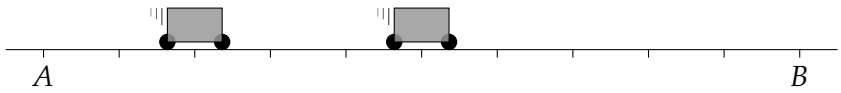
Teilaufgabe 4: Der allgemeine Fall (40 Punkte)

Die Zeitlimite bleibt bestehen und die Gewichtslimite sind nun wieder positive Ganzzahlen.

Förderwagen

Als er klein war, grub Maus Benjamin im Sandkasten ganz viele Tunnel. Kürzlich ist er zum Chef einer Mine in Aserbaidschan ernannt worden. Teil seiner neuen Arbeit ist es, mit Förderwagen das Erz aus den Minen zu befördern. Komplett überfordert von seiner neuen Position bittet er dich um Hilfe.

Es gibt zwei Haltestellen, eine in der Nähe des Abbruchgebietes, Haltestelle A, und eine am Höhlenausgang, Haltestelle B. Beide Haltestellen sind durch ein einziges Gleis miteinander verbunden. Die Strecke kann jeweils nur in eine Richtung gleichzeitig betrieben werden, da die Wagen sonst kollidieren würden. Eine Fahrt auf der Strecke dauert d Sekunden (unabhängig von der Richtung $A \to B$ oder $B \to A$). Maus Benjamin kann jede Sekunde einen neuen Förderwagen auf das Gleis in beliebiger Richtung losschicken.



Am Anfang sind n Förderwagen an der Haltestelle A und 0 an der Haltestelle B. Die Bergläute bauen neue Erzeinheiten zu den Zeiten $t_0, t_1, \ldots, t_{n-1}$ ab, gegeben in aufsteigender Reihenfolge. Einmal abgebaut, kann eine Einheit auf einem Förderwagen von Haltestelle A zu Haltestelle B geschickt zu werden. Ein Förderwagen kann maximal eine Einheit pro Fahrt aufnehmen. Der Arbeitstag von Maus Benjamin ist fertig, sobald alle Erze nach B geschickt wurden und alle Förderwagen wieder zurück an Haltestelle A sind.

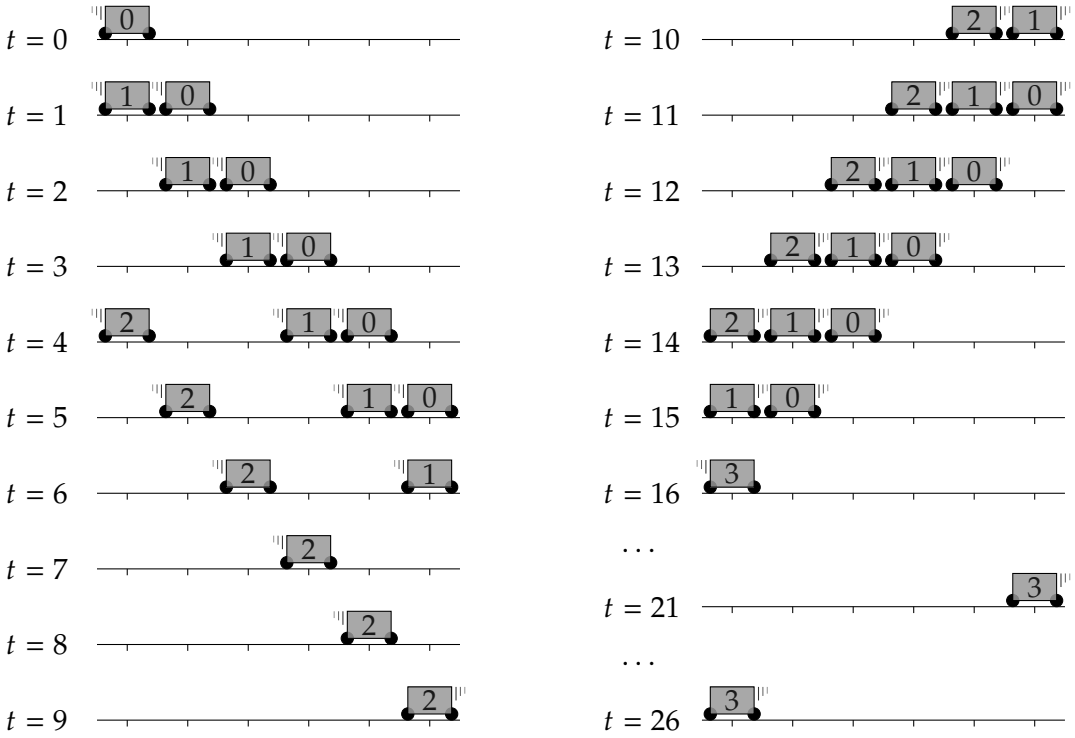
Das Problem von Maus Benjamin ist, die minimale Zeit T herauszufinden, in der es möglich ist, alle Erze zur Haltestelle B zu transportieren und die Förderwagen zurück zur Haltestelle A zu bringen.

Formale Beschreibung Gegeben Ganzzahlen $t_0, t_1, \ldots, t_{n-1}$ mit $t_0 < t_1 < \cdots < t_{n-1}$ und eine Ganzzahl $d > 0$, finde das minimale T, sodass es Ganzzahlen $a_0, a_1, \ldots, a_{n-1}$ (die Zeiten, an denen du den i-ten Förderwagen von A nach B schickst) und $b_0, b_1, \ldots, b_{n-1}$ (die Zeiten, an denen du den i-ten Förderwagen von B nach A schickst) gibt, die Folgendes erfüllen

- $b_i + d \leq T$ für alle i (du bist fertig zum Zeitpunkt T),
- $a_i \geq t_i$ (du kannst nur eine Einheit senden, sobald sie abgebaut wurde),
- $b_i \geq a_i$ (du kannst Förderwagen erst nach A zurücksenden, sobald diese bei B angekommen ist),
- $a_i \neq a_j$ und $b_i \neq b_j$ für $i \neq j$ (keine zwei Wagen starten gleichzeitig) und
- $|a_i - b_j| \geq d$ für alle i und j (du kannst das Gleis nicht in beide Richtungen gleichzeitig bedienen).

Analyse Ein Algorithmus für dieses Problem erhält die beiden ganzen Zahlen n und d, sowie die Liste der Startzeiten $t_0, \ldots, t_{n-1}$, wobei $t_0 < \cdots < t_{n-1}$. Du sollst die Laufzeit und den Speicherverbrauch in Abhängigkeit von n und d angeben. Beide Parameter sind unabhängig voneinander, d.h. $\Theta(n + d)$ ist schlechter als $\Theta(n)$ und $\Theta(d)$. Ähnlich ist $\Theta(\min(n, d))$ besser als $\Theta(n)$ und $\Theta(d)$. Die Werte $t_0, \ldots, t_{n-1}$ zählen nicht zum Speicherverbrauch. Du kannst sie ohne zusätzlichen Speicherverbrauch lesen, aber du darfst sie nicht ändern.

Beispiel Für $n = 4, d = 5$ und $t_0 = 0, t_1 = 1, t_2 = 4, t_3 = 16$, ist die Antwort 26:



Die Startzeiten der Förderwagen sind $a = [0, 1, 4, 21]$ und $b = [11, 10, 9, 26]$.

Teilaufgabe 1: Stofls Algorithmus auffliegen lassen (15 Punkte)

Mouse Stofl hat sich den folgenden Algorithmus ausgedacht, um dieses Problem zu lösen:

In dieser Teilaufgabe musst du:

- Gibt die Laufzeit und den Speicherverbrauch des Stofl-Algorithmus an. Du musst deine Antwort nicht begründen (5 Punkte).
- Finde ein Gegenbeispiel, bei dem der Stofl-Algorithmus keine optimale Lösung liefert. Gib die Ausgabe des Stofl-Algorithmus an (keine Begründung notwendig) und wie die optimale Lösung aussehen würde (10 Punkte).

Algorithm: Stofls Heuristik für Minecarts

input : n, d und das Array $t[0], \ldots, t[n-1]$
output: Eine obere Schranke von T, aber keine minimale

```
1 w = 0                                   // Anzahl Förderwagen, die in B warten
2 for i = 0 to n − 2 do
3 |   x = t[i + 1] − t[i] − 2 · d          // zusätzliche Zeit bei Retour-Fahrt
4 |   if x ≥ 0 then                        // falls Zeit für Retour-Fahrt
5 |   |   w = max(0, w − x)
  |   |    // fahre retour und sende weitere Förderwagen zurück
6 |   else                    // sonst sende das Erz zu B
7 |   |   w = w + 1           // und warte in B
8 return t[n − 1] + 2 · d + w
  // Retour-Fahrt gefolgt von den wartenden Förderwagen
```

Teilaufgabe 2: Optimaler Algorithmus (85 Punkte)

Jetzt bist du dran! Finde einen Algorithmus, der das Problem optimal löst.

Lichtshow

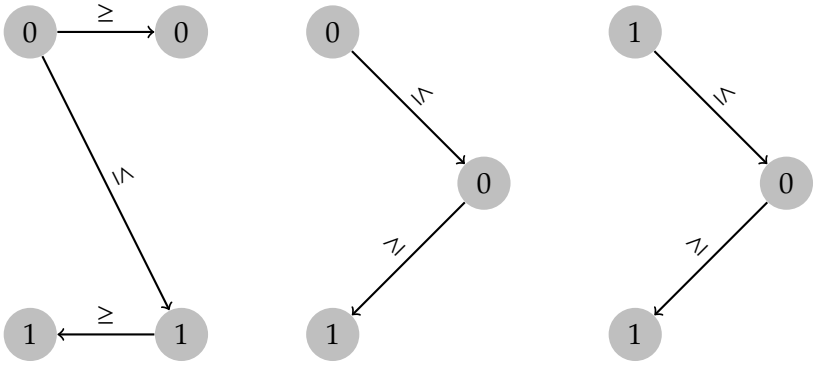
Maus Stofl möchte eine Lichtshow vor seinem Haus veranstalten. Er hat dazu Lichterketten gekauft. Die Lichterketten haben eine seltsame Form: Sie bilden nicht einen langen Streifen, sondern eine Art Baum aus Glühbirnen und Kabeln.

Seine Idee ist es, alle möglichen Kombinationen von aus- und eingeschalteten Lichtern während der Show zu zeigen. Um zwischen den Kombinationen zu wechseln kann Stofl genau eine Glühbirne auf einmal ein- oder ausschalten.

Leider hat er die billigsten Lichterketten aus einem dubiosen Geschäft gekauft und sie sind falsch verdrahtet: Von zwei direkt durch ein Kabel verbundenen Glühbirnen funktioniert genau eine nur dann, wenn auch die andere eingeschaltet ist. Das macht viele Kombinationen unmöglich und enttäuscht Stofl ziemlich. Er hat sich dennoch dazu entschieden, seine Idee durchzuziehen und einfach die unmöglichen Kombinationen zu überspringen. Das macht es jedoch sehr schwierig, eine geeignete Reihenfolge von Kombinationen zu finden. Stofl hat dich gebeten, einen Algorithmus zu entwickeln, der das für ihn erledigt.

Formale Beschreibung Gegeben ist ein Graph G mit n Knoten und m ungerichteten Kanten. Es wird garantiert, dass dieser Graph azyklisch ist, d.h. es gibt keinen Zyklus darin. Der Graph ist also eine Sammlung von Bäumen. Knoten in diesem Graphen haben einen Wert von entweder 0 (aus) oder 1 (an), abhängig vom Zustand der entsprechenden Glühbirne. Kanten sind entweder mit '$\leq$' oder '$\geq$' beschriftet: Wenn eine Kante von Knoten a bis Knoten b mit '$\leq$' beschriftet ist, dann darf der Wert von a nie 1 und der Wert von b 0 sein (und umgekehrt, wenn eine solche Kante mit '$\geq$' beschriftet ist, dann ist es verboten, den Wert von a gleich 0 und den Wert von b gleich 1 zu haben).

Jede Zuordnung der Knoten von G zu $\{0, 1\}$ ist genau dann eine gültige Kombination, wenn alle Bedingungen der Kantenbeschriftungen erfüllt sind. So sind beispielsweise die beiden Graphen links unten eine gültigen Kombination, während das äusserste rechts nicht gültig ist. Deine Aufgabe ist es, einen Algorithmus zu entwickeln, um eine funktionierende Sequenz aller gültigen Kombinationen zu berechnen, so dass sich zwei beliebige aufeinanderfolgende Kombinationen genau in einem Knoten unterscheiden.



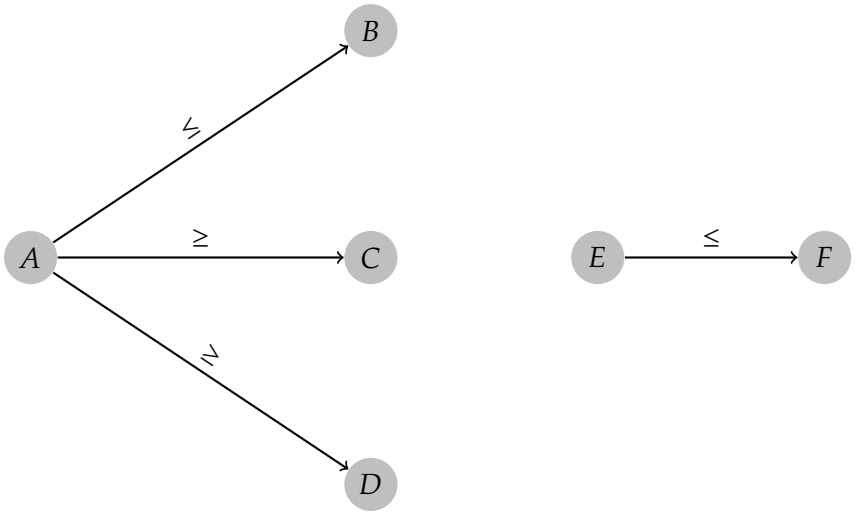
Zu diesem Zweck sollst du die folgenden beiden Funktionen implementieren:

1. *start*: nimmt als Eingabe den Graphen, initialisiert beliebige Datenstrukturen, die du benötigst, und gibt eine Anfangszuweisung von Werten zu Knoten zurück.
2. *next*: gibt den Index des nächsten Knoten zurück, dessen Wert umgeschaltet werden soll oder -1, wenn du das Ende der Sequenz erreicht hast.

Wenn e eine Kante ist, kannst du $e.a$ und $e.b$ verwenden, um zu wissen, welche zwei Knoten durch e verbunden sind. $e.d$ ist '$\leq$', falls a '$\leq$' sein muss, als b und '$\geq$', wenn a '$\geq$' sein muss, als b.

Teilaufgabe 1: Löse das Beispiel (8 Punkte)

Löse das Problem für den untenstehenden Graphen, indem du eine Liste mit dem Anfangszustand jedes Knoten sowie eine Folge von Knoten angibst, welche du wechseln möchtest, so dass jede gültige Kombination genau einmal durchlaufen wird.



Teilaufgabe 2: Kombiniere Lichterketten (24 Punkte)

Zuerst möchte Stofl, dass du ihm hilfst, eine Show mit einigen Lichterketten zu produzieren, für die er bereits eine Lösung gefunden hat: Du musst sie nur kombinieren! In dieser Teilaufgabe kannst du davon ausgehen, dass du für jeden Baum (also jede Zusammenhangskomponente) im Graphen die Funktionen *start* und *next* erhältst. Du bekommst den Graphen als Liste dieser Bäume L. Du kannst Anweisungen wie *L[i].start()* verwenden, um die Funktionen *start* und *next* des i-th-Baums zu verwenden. Deine Aufgabe ist es, die Lichterketten zu kombinieren, mit anderen Worten, du sollst die Funktionen *start* und *next* für den gesamten Graphen implementieren. Du kannst davon ausgehen, dass die Laufzeit der Funktionen für Bäume $O(x)$ für *start* auf einem Baum von x Knoten und $O(1)$ für *next* ist.

Teilaufgabe 3: Einzelne Lichterkette (68 Punkte)

In dieser Teilaufgabe arbeitest du an einer einzelnen zusätzlichen Lichterkette, um die Show zu verbessern. Das ist alles, was du brauchst, denn du kannst sie bereits für Stofl mit deinem Algorithmus aus Subtask 2 kombinieren! Du kannst also davon ausgehen, dass G verbunden ist, d.h. du arbeitest an einem Baum. Du sollst funktionierende *start* und *next* Funktionen für diesen Baum erstellen. Du darfst die folgenden Eigenschaften für diesen Baum annehmen, im Austausch für einen Punktabzug:

Art des Graphs	Maximalpunktzahl
Pfad (maximal Grad 2) mit Kanten von i bis $i+1$ mit Becshriftung '$\geq$'	8
Binärbaum (maximal Grad 3), von dem du die Wurzel r erhältst. Kanten sind so beschriftet, dass der Elternteil immer '$\geq$' seinem Kind ist.	24
Baum, von dem du die Wurzel r erhältst. Kanten sind so beschriftet, dass der Elternteil immer '$\geq$' seinem Kind ist.	34
Baum mit beliebiger Beschriftung	68

Bewertung In den beiden letzten Teilaufgaben liegt der Schwerpunkt auf der Korrektheit deines Algorithmus. Dein Algorithmus muss nicht optimal sein, um einige oder sogar die meisten Punkte zu erhalten. Hier ist eine Liste der Laufzeiten, die du anstreben kannst, und die Prozentsätze der Gesamtpunktzahl, die du für jeden von ihnen erhalten kannst. In dieser Tabelle steht P für die Anzahl der gültigen Kombinationen und $\text{poly}(n)$ für ein Polynom in n.

Laufzeit	Anteil der Punkte
start in $O(P \cdot \text{poly}(n))$, *next* in konstanter Zeit	25%
start in $O(\text{poly}(n))$, *next* in $O(n)$	50%
start in $O(n)$, *next* in konstanter Zeit im Durchschnittsfall	75%
start in $O(n)$, *next* in konstanter Zeit im schlimmsten Fall	100%