Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Übungsblatt 12

Algorithmen I - Sommersemester 2024

Abgabe im ILIAS bis 19.07.2024, 18:00 Uhr

Die Abgabe erfolgt alleine oder zu zweit als *eine* PDF-Datei über das Übungsmodul in der Gruppe eures Tutoriums im ILIAS. Beschriftet die Abgabe deutlich mit Matrikelnummer und Name.

- Achtet bei handschriftlichen Abgaben auf Lesbarkeit.
- Achtet darauf, ob Algorithmen in Worten oder Pseudocode beschrieben werden sollen.
- Es werden immer asymptotisch möglichst effiziente Algorithmen erwartet, wenn nicht anders angegeben.
- Wenn Korrektheits- oder Laufzeitanalysen gefordert sind, behandelt diese separat von der Algorithmenbeschreibung.

Gesamtpunkte: 20 (+ 3 Bonus)

Aufgabe 1 - Zu schwer - Gekreissche auf der Waage (3 Punkte)

Sei $G = (V, E)$ ein zusammenhängender, gewichteter Graph, bei dem keine zwei Kanten das gleiche Gewicht haben. Sei $C \subseteq V$ ein Kreis in G. Beweise, dass es keinen minimalen Spannbaum von G gibt, der die schwerste Kante von C enthält.

Lösung 1

Sei $T = (V, E_T)$ ein minimaler Spannbaum von G. Wir bezeichnen die schwerste Kante in C als $e_{\max} = \{u, v\}$. Angenommen, es wäre $e_{\max} \in E_T$. Wenn wir $e_{\max}$ aus T entfernen, so zerfällt T in zwei Zusammenhangskomponenten A und B, sodass u und v nicht in der gleichen Zusammenhangskomponente liegen. Betrachte nun den Pfad P von u nach v der entsteht, wenn $\{u, v\}$ aus C entfernt wird. Da u und v in unterschiedlichen Komponenten liegen, gibt es mindestens eine Kante $e = \{u', v'\}$ auf P, sodass $u' \in A$ und $v' \in B$. Durch Hinzufügen von e erhalten wir $T' = (V, E'_T)$ mit $E'_T = (E_T \backslash \{e_{\max}\}) \cup \{e\}$. Dabei ist T' ein Baum, denn $|E_T| = n - 1$ und somit $|E'_T| = n - 1$ und durch Hinzunahme von e werden die beiden Komponenten A und B zu einer Zusammenhangskomponente verbunden, d.h. T' ist zusammenhängend und hat $n - 1$ Kanten. Da $e_{\max}$ die schwerste Kante auf

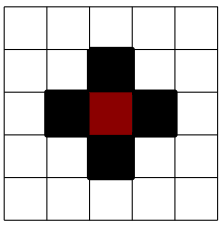
C ist und alle Kantengewichte verschieden sind, ist e leichter als $e_{\max}$. Damit ist T' ein Spannbaum von G, dessen Gesamtgewicht der Kanten kleiner als das der Kanten in T ist. $\lightning$ Widerspruch zu: T ist minimaler Spannbaum von G.

Aufgabe 2 - Biber aller Gangs, vereinigt euch! (8 Punkte)

In ganz Deutschland finden sich in jeder Ecke Biber. Durchlöcherte Dämme, durchgenagte Strommasten und fehlende Holzbalken, die Biberplage geht an keinem vorbei. Man munkelt, dass das sogar organisierte Verbrechen von Biber-Gangs sein könnten! Diesem Verdacht möchte das BaFin (**Bi**ber**a**ufsichts- und **F**orschungs**in**stitut) nachgehen. Daher teilt das BaFin Deutschland mithilfe eines quadratischen Gitternetzes in verschiedene Bereiche ein. Als erstes konnten die Angestellten des BaFin herausfinden, in welchen Gitterzellen die Biber bereits ihr Unwesen treiben. Dabei haben sie schon Folgendes über die Biber herausgefunden:

- In jeder Zelle ist höchstens ein Biber. Eine Matrix gibt an, ob sich in einer Zelle ein Biber befindet oder nicht.
- Biber können mit Bibern in benachbarten (oben, unten, links, rechts) Zellen direkt kommunizieren. Indirekt kann ein Biber mit einem anderem Biber kommunizieren, wenn es möglich ist, eine Nachricht durch direkte Kommunikationen durchzugeben.
- Zwei Biber, die miteinander kommunizieren können, sind in der gleichen Gang, sonst in verschiedenen.

Als nächstes sollen nun die Analysten bestimmen, welcher Biber welcher Gang angehört, damit diese erfolgreich zerschlagen werden kann. Jede Gang bekommt dabei eine eindeutige Nummer zugeordnet. Wir stellen nun das Gitter als ein 2D-Array dar. Eine Zelle enthält genau dann ein x, wenn ein Biber in dieser Zelle ist.



	x							
x	x	x					x	
	x	x	x					
				x	x			
		x		x	x			
	x	x						
	x			x	x	x		
	x				x	x	x	
						x	x	x

Links ist die Nachbarschaft der roten Zelle in schwarz dargestellt. Rechts sind die Zellen eines Gitters gegeben. Zellen mit einem x enthalten einen Biber, die verbleibenden Zellen sind biberfrei.

a) Gib für das Gitter und die Biberverteilung des obigen Beispiels eine Zuweisung der Biber zu Gangs an. (1 Punkt)

Wir wollen nun einen Algorithmus entwickeln, der für jeden Biber seine Gang bestimmt. Dazu durchlaufen wir das Gitter, indem wir Zeilen von oben nach unten und innerhalb einer Zeile die Zellen von links nach rechts betrachten. Wir nehmen im Folgenden an, dass wir aktuell die Zelle (i, j) betrachten und dass diese einen Biber enthält. Von den vier Nachbarn von Zelle (i, j) haben wir bisher also nur den linken und den oberen Nachbar betrachtet. Wir wollen uns nun überlegen, unter welchen Bedingungen wir den Biber welcher Gang zuweisen müssen.

Wenn die linke und die obere Zelle biberfrei sind, weisen wir den Biber in Zelle (i, j) einer neuen Gang zu (siehe gelb markierte Zelle im Bild).

b) Welcher Gang weisen wir dem Biber in Zelle (i, j) zu, wenn genau eine der beiden Zellen direkt links oder direkt oben einen Biber enthält? (1 Punkt)

c) Gib die Bedingung an, unter der sich bei Betrachtung von Zelle (i, j) die Gangs von bereits gesehenen Biber ändert. Welcher Gang wird dann der Biber in Zelle (i, j) zugewiesen? (1 Punkt)

d) Gib ein Beispiel (mit mindestens 7 Bibern) an, in dem bei der Abarbeitung einer einzelnen Zelle die Hälfte der bisher zugewiesenen Biber einer neuen Gang zugewiesen werden. Gib hierfür ein Gitternetz an, in dem du markiert hast, wo sich Biber befinden. Markiere außerdem die Zelle, bei deren Bearbeitung die Änderung auftritt und die Zellen, die einer neuen Gang zugewiesen werden müssen. (1 Punkt)

Nun wollen wir diese lokalen Erkenntnisse nutzen, um einen globalen Algorithmus anzugeben. Hierfür nutzen wir eine dynamische Variante der in Vorlesung 17 vorgestellten Union-Find-Datenstruktur. Es gibt nun zusätzlich eine Methode `insert`(x), die x als einzelnes Element mit Zeiger auf sich selbst in den Wald einfügt.

e) Gib nun einen Algorithmus in Worten an, der das Gitter wie oben beschrieben abläuft und jeden Biber einer Gang zuordnet. Der Algorithmus soll dann für jede Gang alle Biber ausgeben, die zu dieser Gang gehören. Die Laufzeit soll für ein Gitter mit $\sqrt{n}$ Zeilen und $\sqrt{n}$ Spalten in $\mathcal{O}(n \cdot \log^*(n))$ sein. Begründe außerdem die Laufzeit deines Algorithmus. (4 Punkte)

Hinweis: Überlege dir, welche Fälle für die linke und obere Nachbarzelle auftreten können, und was in diesen Fällen zu tun ist. Übertrage dies dann auf die Operationen der Datenstruktur.

Lösung 2

Diese Aufgabe setzt den Hoshen-Kopelman Algorithmus um `https://en.wikipedia.org/wiki/Hoshen%E2%80%93Kopelman_algorithm`.

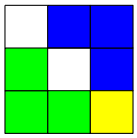
a) Das untenstehende Gitter zeigt die Lösung.

	1							
1	1	1					3	
	1	1	1					
				2	2			
		4		2	2			
	4	4						
	4			5	5	5		
	4				5	5	5	
						5	5	5

b) Hier besteht eine Verbindung zu genau einer bereits existierenden Gang. Somit weisen wir die Zelle dieser Gang zu.

c) Wenn sowohl die linke als auch die obere Nachbarzelle Biber aus verschiedenen Gangs enthalten, werden die beiden Gangs durch die betrachtete Zelle zu einer großen Gang vereinigt. Dies ist dadurch begründet, dass es nun eine Verbindung zwischen den Bibern der Gangs gibt. Somit vereinigen wir die beiden Gangs und weisen den Biber aus Zelle (i, j) der vereinigten Gang zu.

d) Im untenstehende Beispiel bilden, bis zur Abarbeitung der gelben Zelle, die Biber der grünen und blauen Zellen jeweils eine Gang. Nach der Abarbeitung bilden die Biber aller 7 Zellen eine gemeinsame Gang und wir müssen z. B. alle blauen Zellen umfärben.



e) Wir verwalten die Gangs in einer Union-Find-Datenstruktur. Hierbei sind die Zellen die gespeicherten Elemente. Wir beginnen mit einer leeren Union-Find-Datenstruktur. Nun gehen wir zeilenweise durch das Gitternetz und arbeiten jeweils die betrachtete Zelle ab. Hier gehen wir wie in den bisherigen Aufgabenteilen argumentiert vor. Insbesondere fügen wir also Zellen, deren linker und oberer Nachbar biberfrei, sind einer neuen Gang hinzu und bestimmen die Nummer dieser Gang dadurch, dass wir uns merken, wie viele Gangs es bereits gibt. Wir fügen anschließend diese Zelle in die Datenstruktur ein. Hat eine Zelle genau einen Biber als Nachbar, weisen wir ihr die Nummer einer existierenden Gang zu. Das heißt, dass wir diese Zelle in die Datenstruktur einfügen und diese Zelle mit der Gang ihres Nachbarn vereinigen. Für eine Zelle mit zwei Nachbarn vereinigen wir zuerst die Gangs der Nachbarn und gehen dann wie bei einer Zelle mit einem Nachbarn vor. Zellen, die keine Biber enthalten werden übersprungen. Abschließend iterieren wir einmal über alle Zellen und bestimmen für jede Zelle mittels `find` die Nummer der Gangs, die wir in die Matrix eintragen müssen.

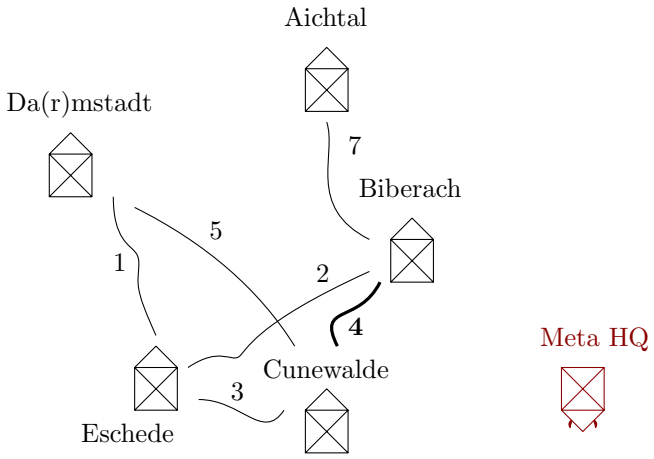
Wir iterieren zwei Mal über alle n Werte und rufen jeweils konstant viele Operationen auf der Union-Find-Datenstruktur auf. Somit erfüllen wir die geforderte Laufzeit.

Aufgabe 3 - Meta-Analyse (9 Punkte)

Um Dr. Meta endlich aufhalten zu können und seine Biber zu befreien, expandiert das Biber-Schutz-Institut (BSI)[1]. Neben ihrer Zentrale in Rotterdamm sollen nun viele weitere Stützpunkte eröffnet werden.

Jeder Stützpunkt soll mit jedem anderen Stützpunkt kommunizieren können, gegebenenfalls über den Umweg über andere Stützpunkte. Dafür möchte das BSI nun jeweils zwei Stützpunkte miteinander durch einen geheimen Kommunikationskanal verbinden. Für jedes Paar von Stützpunkten weiß das BSI, wie teuer der Ausbau eines Kanals wäre. Die Kosten sind immer strikt positiv. Für manche Paare ist der Kanal so teuer, dass dieser auf gar keinen Fall gebaut wird. Umgekehrt ist für manche Paare die direkte Kommunikation besonders wichtig, weshalb der Kanal zwischen diesen unbedingt gebaut werden muss.[2] In beiden Fällen sind dem BSI diese Paare bekannt.

Zwischen jedem Paar aus Stützpunkten soll es genau einen Weg der Kommunikation geben. Außerdem sollen die Gesamtkosten möglichst gering sein[3].



Der Lageplan des BSI zeigt einen möglichen Ausbau. In schwarz sind fünf mögliche Außenstellen gegeben. Die möglichen Kommunikationskanäle sind mit den jeweiligen Ausbaukosten beschriftet. Kommunikationskanäle, deren Ausbau zu teuer sind, sind nicht eingezeichnet. Wichtige Kommunikationskanäle, die ausgebaut werden müssen, sind fett markiert.

a) Markiere in dem obigen Beispiel alle Kommunikationskanäle, die das BSI entsprechend der obigen Anforderungen ausbauen sollte. Wie viel kostet der Ausbau dann

[1] Diese und die nächsten beiden Fußnoten geben zusätzliche Informationen, die die Expansion motivieren. Die darin gegebenen Informationen sind für die Aufgabe nicht relevant.

[2] Hier beaufsichtigen die beiden Stützpunkte gemeinsam ein Geheimversteck von Dr. Meta und müssen daher direkt miteinander kommunizieren können.

[3] Das BSI möchte schließlich seine begrenzten Mittel nicht vollständig für den Ausbau der Anlagen aufwenden.

insgesamt? (0.5 Punkte)

b) Erkläre, wie du das Problem mithilfe eines Graphen modellieren kannst. Gehe dabei insbesondere darauf ein, wie sich die Problemstellung (Minimierung der Ausbaukosten unter den gegebenen Voraussetzungen) auf den Graphen überträgt. (1.5 Punkte)

Zunächst interessiert uns, ob es überhaupt möglich ist, Kommunikationskanäle so auszubauen, sodass alle Anforderungen erfüllt sind. [4] Hierfür ignorieren wir vorerst die Kosten des Ausbaus.

c) Wir nehmen an, dass es genau **ein** Paar gibt, das direkt miteinander kommunizieren muss. Gib an, unter welchen Bedingungen es in diesem Fall eine Lösung gibt. Beweise deine Antwort. (1.5 Punkte)

d) Nun kann es **beliebig viele** Paare geben, die direkt miteinander kommunizieren müssen. Gib an, unter welchen Bedingungen es in diesem Fall eine Lösung gibt. Beweise deine Antwort. (1.5 Punkte)

Wir wollen nun eine minimale Lösung des Problems finden und dafür einen Algorithmus angeben. Dazu nehmen wir an, dass die Bedingungen aus c) und d) erfüllt sind, es also stets eine Lösung gibt. Beachte, dass wir nun wieder an der Minimalität interessiert sind, die Kosten der Kommunikationskanäle also wichtig sind.

e) Wir verwenden zuerst **Prims** Algorithmus, um eine minimale Lösung zu finden. Beschreibe, wie der Graph modifiziert wird, sodass Prims Algorithmus das Geforderte ausgibt. In dieser Teilaufgabe darf *der Algorithmus* selbst nicht angepasst werden. Begründe die Laufzeit und Korrektheit deines Algorithmus. (2 Punkte)

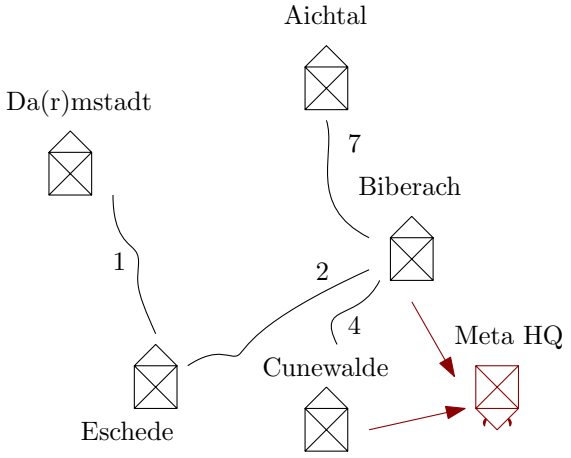
Hinweis: Für die Laufzeit musst du nur argumentieren, dass deine Anpassungen Prims Algorithmus nicht verlangsamen. Für die Korrektheit darfst du ebenfalls auf der Korrektheit von Prims Algorithmus aufbauen. Selbiges gilt für Kruskals Algorithmus in der nächsten Teilaufgabe.

f) Jetzt verwenden wir **Kruskals** Algorithmus, um eine minimale Lösung zu finden. Beschreibe dazu, wie Kruskals Algorithmus angepasst wird, sodass Kruskals Algorithmus das Geforderte ausgibt. In dieser Teilaufgabe darf *der Graph* nicht angepasst werden. Begründe die Laufzeit und Korrektheit deines Algorithmus. (2 Punkte)

Lösung 3

a) Die untenstehende Graphik zeigt die Kommunikationskanäle, die ausgebaut werden sollen.

[4] Insbesondere die Forderung, dass es nur einen Weg der Kommunikation zwischen jedem Stützpunktpaar gibt.



b) Wir modellieren das Kommunikationsnetz als ungerichteten und gewichteten Graphen. Hierfür modellieren wir die Stützpunkte als Knoten und verbinden zwei Stützpunkte, falls die Anlage zwischen den beiden nicht schon von vorneherein als zu teuer bestimmt wurde. Als Kantengewichte nutzen wir die Installationskosten.

   Die Problemstellung können wir wie folgt auf unseren Graphen übertragen: Wir suchen einen Minimalen Spannbaum des Graphen, wobei bestimme Kanten bereits vorgegeben sind.

c) Da Spannbäume zusammenhängende Subgraphen sind, muss der Graph zusammenhängend sein, damit wir eine Lösung, also einen Spannbaum mit vorgegebenen Kanten, finden können. Wir zeigen nun, dass diese Bedingung für den Fall mit einer Kante ausreichend ist. Hierfür sei e die vorgegebene Kante, die unser Spannbaum enthalten soll. Wir betrachten nun einen beliebigen Spannbaum T. Ein solcher exisitert, da der Graph zusammenhängend ist. Falls T die Kante e bereits enthält, sind wir fertig. Anderenfalls können wir e zu T hinzufügen, wodurch T nun einen Kreis C enthält. Hierbei ist e ein Teil von C. Wir können nun eine Kante $e' \in C$ mit $e' \neq e$ aus T entfernen und erhalten somit einen Spannbaum, der e enthält.

d) Wir orientieren uns an der Lösung der letzten Teilaufgabe und passen diese an. Der zugrundeliegende Graph muss weiterhin zusammenhängend sein. Wir betrachten nun die Menge S der vorgegebenen Kanten. Offensichtlicherweise muss diese Menge kreisfrei sein, es darf also keine Teilmenge von S geben, die ein Kreis im Graphen bildet. Dies ist nötig, da der kreisfreie Spannbaum diese Menge enthalten muss. Diese beiden offensichtlicherweise notwendigen Kriterien sind auch bereits ausreichend.

   Um dies zu zeigen können wir analog zur letzten Teilaufgabe vorgehen. Wir starten mit einem beliebigen Spannbaum T und fügen iterativ eine der verbeibenden Kanten aus S in den Baum T ein. Somit entsteht ein Kreis C und wir entfernen dann eine Kante aus $C \setminus S$. Da S kreisfrei ist, gibt es stets eine solche Kante.

e) Wir bezeichnen erneut die Menge der Kanten, die vorgegeben sind mit S. Wir setzen nun die Kosten aller Kanten aus S auf 0 und führen Prims Algorithmus

aus. Abschließend rekonstruieren wir die Kosten des Spannbaums mithilfe der ursprünglichen Kantegewichte. Da wir nur die Metrik anpassen ändert sich die Laufzeit von Prims Algorithmus nicht. Diese ist somit $\Theta((n+m) \cdot \log(n))$ mit binären Heaps.

Die Korrektheit auf dem modifizierten Graphen folgt aus der Korrektheit des Algorithmus von Prim. Es bleibt zu zeigen, dass die Anpassungen korrekt sind. Zum einen werden alle Kanten aus S durch den Algorithmus von Prim ausgewählt, da diese Gewicht 0 erhalten und somit stets leichter als alle anderen Kanten, die strikt positives Gewicht haben, sind. Hier ist die Schnitteigenschaft relevant. Somit findet der Algorithmus eine gültige Lösung. Zum anderen ist es für einen Spannbaum T ausreichend die Kanten $T \setminus S$ zu minimieren, um T zu minimieren. Dies ist durch die Tatsache begründet, dass alle gültigen Spannbäume die Kanten aus S enthalten müssen. Somit findet der Algorithmus auch eine minimale Lösung.

f) Wir bezeichnen erneut die Menge der Kanten, die vorgegeben sind mit S. Wir rufen nun für jede Kante $uv \in S$ die Operation `union`(u, v) auf unserer Union-Find-Datenstruktur auf und führen anschließend Kruskals Algorithmus aus. Der Aufruf der `union`-Operationen benötigt $|S| \cdot \Theta(\log^*(n)) = \Theta(|S| \cdot \log^*(n)) \subseteq \Theta(n \cdot \log^*(n))$ Zeit. Somit wird die Laufzeit durch die Laufzeit von Kruskals Algorithmus dominiert.

Auch hier nutzen wir die Korrektheit von Kruskals Algorithmus und zeigen, dass unsere Anpassungen korrekt sind. Durch die initialen Operationen sind die Kanten aus S bereits ausgewählt und somit im Spannbaum enthalten. Wie in der letzten Teilaufgabe reicht es ebenfalls, die verbleibenden Kanten zu minimieren. Somit finden wir auch eine minimale Lösung.

Aufgabe 4 - Zum Knobeln (0 Punkte + 3 Bonus)

Déjà-vu - Für die letzte Aufgabe des vorletzten Blattes haben wir nochmal die erste Aufgabe des ersten Blattes mitgebracht.

Ordne die folgenden Funktionen so an, dass $f \in \mathcal{O}(g)$ gilt, genau dann, wenn f links von g eingeordnet ist. Begründe die Anordnung der Funktionen. Wir nutzen hierbei den Logarithmus zur Basis e. (3 Punkte)

(i) Ackermannfunktion $a(n, n)$ nach der Definition von Peter

(ii) Busy Beaver Funktion $\Sigma(n)$[5]

(iii) $\log(n)^{(\log(n)^n)!}$

(iv) Potenzturm P_n aus Zweien der Höhe n

(v) $e^{(n^n)!}$

[5] `https://en.wikipedia.org/wiki/Busy_beaver#The_busy_beaver_function_%CE%A3`

Lösung 4

Die richtige Reihenfolge ist: (iii), (v), (iv), (i), (ii)
Also:

$$\log(n)^{(\log(n)^n)!}, \quad e^{(n^n)!}, \quad P_n, \quad a(n,n), \quad \Sigma(n)$$

Begründungen.

- Es gilt

$$\begin{aligned}
\log(n)^{(\log(n)^n)!} &\leq n^{(\log(n)^n)!} \\
&= e^{\log(n)\cdot(\log(n)^n)!} \\
&\leq e^{(\log(n)^n+1)\cdot(\log(n)^n)!} \\
&= e^{(\log(n)^n+1)!} \\
&\leq e^{(n^n)!}
\end{aligned}$$

  Wobei wir im letzten Schritt $\log(n)^n + 1 \leq n^n$ für ausreichend große n benutzt haben.

- Für $e^{(n^n)!}$ beachten wir zuerst, dass nach Vorlesung n durch einen Potenzturm aus Zweien der Höhe $\log^*(n)$ dargestellt werden kann. Anschließend nutzen wir die folgende Abschätzung:

$$\begin{aligned}
e^{(n^n)!} &\leq 4^{(n^n)^{(n^n)}} \\
&\leq 4^{n^{n^{n^n}}} \\
&= (2^2)^{n^{n^{n^n}}}
\end{aligned}$$

  Wenn wir nun jedes n durch einen Potenzturm aus Zweien ersetzen, haben wir nur $2 + 4 \cdot \log^*(n)$ Zweien verwendet. Für ausreichend große n wächst somit (iv) schneller als (v).

- In der Vorlesung haben wir gesehen, dass $\alpha(n)$ langsamer wächst als $\log^*(n)$. Hier betrachten wir ihre Inversen.

- $\Sigma(n)$ kann durch eine Turingmaschine nicht berechnet werden und wächst daher schneller als die anderen, berechenbaren Funktionen.

Eulenfest am 12. Juli 2024

Nächste Woche Freitag findet endlich wieder das Eulenfest statt! Los geht es um 18:30 Uhr, im und um den Infobau. Dieses Jahr gibt es auf der großen Bühne Livemusik von den Bands *JxP* und *Canalopi*. Danach legen unsere DJs *Paul Städter*, *Julicorn und Luis Reissenweber* sowie *queerbeats* auf. Natürlich gibt es auch wieder Karaoke. Außerdem werdet ihr mit Getränken und Essen versorgt. Weitere Infos zum Fest findet ihr unter eulen.party.

Wir suchen auch noch tatkräftige Unterstützung für unser Helferteam. Ob beim Auf- oder Abbau, als Security oder als Betreuer für die zahlreichen Essens- und Getränkestände. Für die Helfer bleibt dabei auch immer was an den Ständen übrig. Tragt euch dafür einfach ins Helfersystem (redseat.de/eulenfest2024) ein.

Mehr Infos und Helfersystem