Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prüfung **(Lösung)**

Datenstrukturen und Algorithmen (D-MATH RW)

Felix Friedrich, Departement Informatik

ETH Zürich, 9.2.2019.

Name, Vorname: . . . . .
Legi-Nummer: . . . . .

Ich bestätige mit meiner Unterschrift, dass ich diese Prüfung unter regulären Bedingungen ablegen konnte und dass ich die allgemeinen Richtlinien gelesen und verstanden habe.

I confirm with my signature that I was able to take this exam under regular conditions and that I have read and understood the general guidelines.

Unterschrift:

Allgemeine Richtlinien:

1. Dauer der Prüfung: 120 Minuten.
2. Erlaubte Unterlagen: Wörterbuch (für von Ihnen gesprochene Sprachen). 4 A4 Seiten handgeschrieben oder $\geq$ 11pt Schriftgrösse.
3. Benützen Sie einen Kugelschreiber (blau oder schwarz) und keinen Bleistift. Bitte schreiben Sie leserlich. Nur lesbare Resultate werden bewertet.
4. Lösungen sind direkt auf das Aufgabenblatt in die dafür vorgesehenen Boxen zu schreiben (und direkt darunter, falls mehr Platz benötigt wird). Ungültige Lösungen sind deutlich durchzustreichen! Korrekturen bei Multiple-Choice Aufgaben bitte unmissverständlich anbringen!
5. Es gibt keine Negativpunkte für falsche Antworten.
6. Störungen durch irgendjemanden oder irgendetwas melden Sie bitte sofort der Aufsichtsperson.
7. Wir sammeln die Prüfung zum Schluss ein. Wichtig: Stellen Sie unbedingt selbst sicher, dass Ihre Prüfung von einem Assistenten eingezogen wird. Stecken Sie keine Prüfung ein und lassen Sie Ihre Prüfung nicht einfach am Platz liegen. Dasselbe gilt, wenn Sie früher abgeben wollen: Bitte melden Sie sich lautlos, und wir holen die Prüfung ab. Vorzeitige Abgaben sind nur bis 15 Minuten vor Prüfungsende möglich.
8. Wenn Sie zur Toilette müssen, melden Sie dies einer Aufsichtsperson durch Handzeichen.
9. Wir beantworten keine inhaltlichen Fragen während der Prüfung. Kommentare zur Aufgabe schreiben Sie bitte auf das Aufgabenblatt.

General guidelines:

1. *Exam duration: 120 minutes.*
2. *Permitted examination aids: dictionary (for languages spoken by yourself). 4 A4 pages hand written or $\geq$ 11pt font size.*
3. *Use a pen (black or blue), not a pencil. Please write legibly. We will only consider solutions that we can read.*
4. *Solutions must be written directly onto the exam sheets in the provided boxes (and directly below, if more space is needed). Invalid solutions need to be crossed out clearly. Provide corrections to answers of multiple choice questions without any ambiguity!*
5. *There are no negative points for wrong answers.*
6. *If you feel disturbed by anyone or anything, let the supervisor of the exam know immediately.*
7. *We collect the exams at the end. Important: You must ensure that your exam has been collected by an assistant. Do not take any exam with you and do not leave your exam behind on your desk. The same applies when you want to finish early: Please contact us silently and we will collect the exam. Handing in your exam ahead of time is only possible until 15 minutes before the exam ends.*
8. *If you need to go to the toilet, raise your hand and wait for a supervisor.*
9. *We will not answer any content-related questions during the exam. Please write comments referring to the tasks on the exam sheets.*

Question:	1	2	3	4	5	6	7	Total
Points:	20	16	18	17	10	17	10	108
Score:								

Generelle Anmerkung / *General Remark*

Verwenden Sie die Notation, Algorithmen und Datenstrukturen aus der Vorlesung. Falls Sie andere Methoden verwenden, müssen Sie diese kurz so erklären, dass Ihre Ergebnisse nachvollziehbar sind.

Use notation, algorithms and data structures from the course. If you use different methods, you need to explain them such that your results are reproducible.

Aufgabe 1: Verschiedenes (20P)

1) In dieser Aufgabe sollen nur Ergebnisse angegeben werden. Begründungen sind nicht notwendig.

   In this task only results have to be provided. Explanations are not required.

2) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

   As the order of characters we take the alphabetical order and for numbers we take the ascending order according to their sizes.

/2P (a) Führen Sie auf folgenden Array zwei Iterationen des Algorithmus natürlicher 2-Wege Mergesort aus. Tipp: der natürliche Algorithmus nutzt aus, dass das Array in Teilen vorsortiert ist.

Execute on the following array two iterations of the algorithm Natural 2-way Mergesort. Hint: the natural algorithm makes use of the previously sorted parts of the array.

2	3	8	5	7	9	6	4	10
0	1	2	3	4	5	6	7	8
2	3	5	7	8	9	4	6	10
0	1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8

/2P (b) Nennen Sie einen wichtigen Nachteil des Quicksort-Algorithmus im Vergleich zum Mergesort Algorithmus.

Name an important disadvantage of the Quicksort algorithm, when compared to Mergesort.

Only expected worst-case running time $n \log n$

Nennen Sie einen wichtigen Nachteil des Mergesort-Algorithmus im Vergleich zum Quicksort Algorithmus.

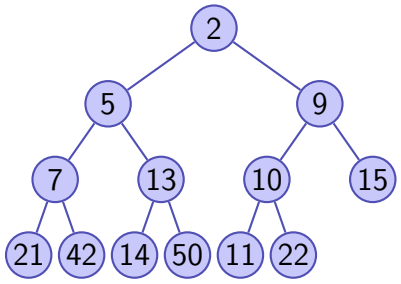
Name an important disadvantage of the Mergesort-Algorithm, compared to Quicksort.

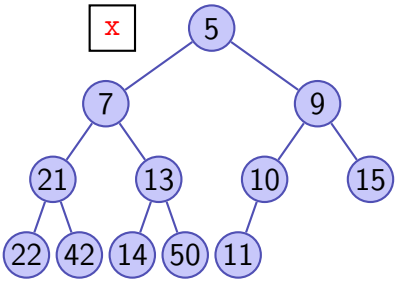
Need intermediate storage for the merging step.

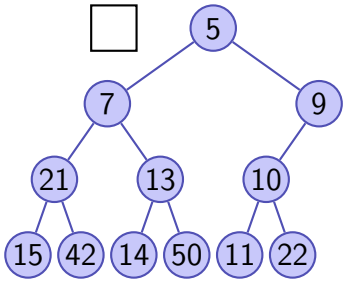
/2P

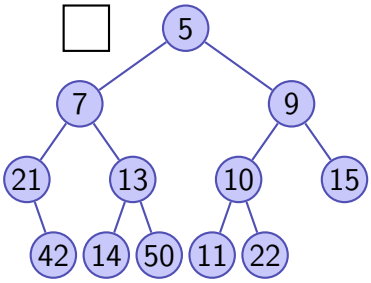
(c) Führen Sie auf folgendem Min-Heap eine Extract-Min Operation aus, wie in der Vorlesung vorgestellt, einschliesslich der Wiederherstellung der Heap-Bedingung. Wie sieht der Heap nach der Operation aus? Kreuzen Sie die richtige Antwort an.

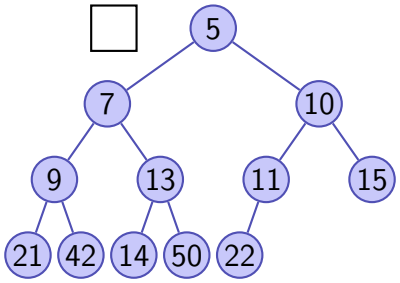
On the following Min-Heap, perform an extract-min operation, including re-establishing the heap-condition, as shown in class. What does the heap look like after the operation? Mark the correct answer.

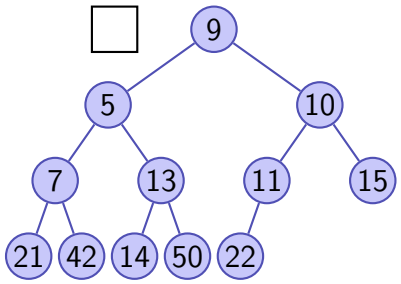


[x]


[ ]


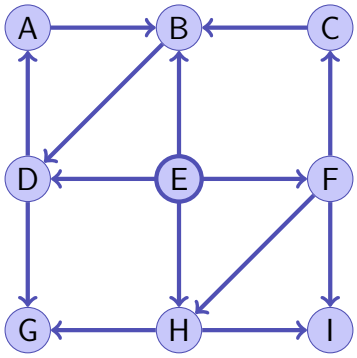
[ ]


[ ]


[ ]


/2P (d) Besuchen Sie folgenden Graphen mit einer Breitensuche und einer Tiefensuche ausgehend vom Knoten E. Wenn es mehrere Möglichkeiten für eine Besuchsreihenfolge der Nachbarn gibt, wird immer die alphabetische Ordnung genommen.

Visit the following graph with a breadth-first search and a depth-first search starting with node E. If there are several possibilities for a visiting order of the neighbouts, choose the alphabetical order.



Breitensuche/*Breadth First Search*:

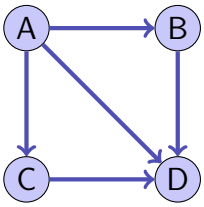
E B D F H A G C I

Tiefensuche/*Depth First Search*:

E B D A G F C H I

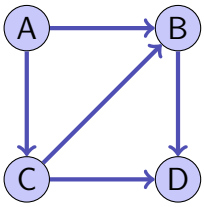
/3P (e) Auf wie viele Arten können die folgenden gerichteten Graphen jeweils topologisch sortiert werden?

In how many ways can the following directed graphs be topologically sorted each?



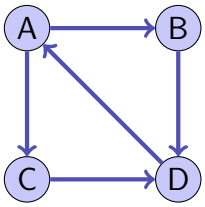
Anzahl Sortierungen / *number sortings*

2



Anzahl Sortierungen / *number sortings*

1

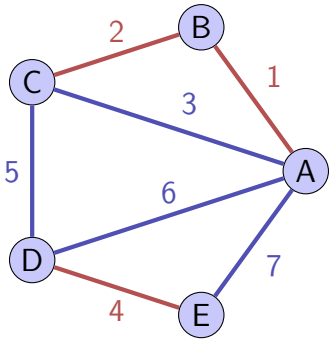


Anzahl Sortierungen / *number sortings*

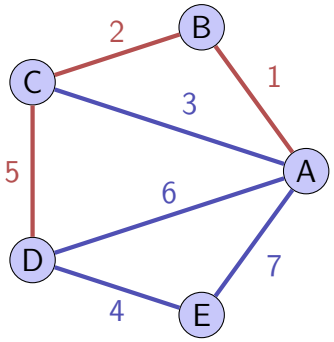
0

/4P (f) Nachfolgend ist derselbe Graph zweimal abgebildet. Markieren Sie im linken Graphen die drei Kanten, die der Algorithmus von Kruskal bei der Berechnung des minimalen Spannbaums zuerst hinzunimmt. Markieren Sie im rechten Graphen die drei Kanten, die der Algorithmus von Prim/Dijkstra bei der Berechnung des minimalen Spannbaums zuerst hinzunimmt, wenn er bei A startet.

Below the same graph is shown twice. Mark in the left graph the three edges that Kruskal's algorithm adds first when deriving a minimum spanning tree.
Mark in the right graph the three nodes that Prim/Dijkstra's algorithm adds first when deriving a minimum spanning tree, provided it starts at node A-



(Kruskal)

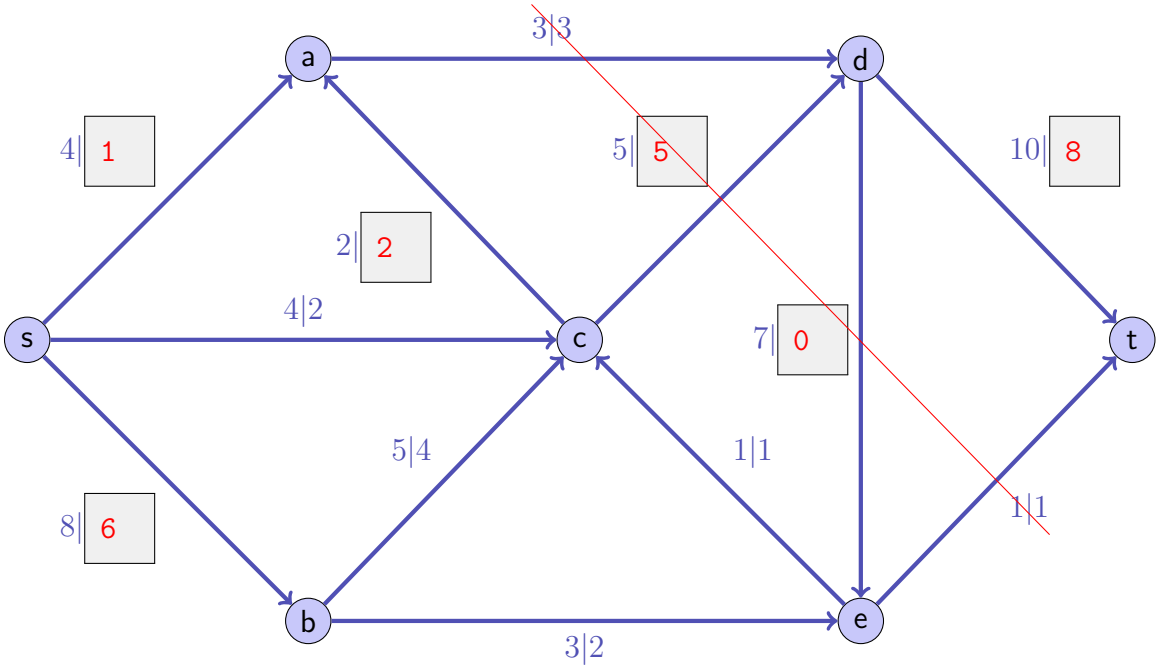


(Prim/Dijkstra)

(g) Gegeben ist das folgende Flussnetzwerk mit Quelle s und Senke t. Die einzelnen Kapazitäten c_i und Flüsse ϕ_i sind an den Kanten angegeben als $c_i|\phi_i$. Ergänzen Sie die fehlenden Flusswerte auf den Kanten, so dass ϕ ein gültiger Fluss ist. Dieser wird (in diesem Beispiel) in jedem Falle maximal sein. Zeichnen Sie in der Abbildung einen Schnitt ein, der zeigt, dass ϕ maximal ist.

Provided in the following is a flow network with source s and sink t. Capacities c_i and flows ϕ_i are provided at the edges as $c_i|\phi_i$. Complete the missing flow values at the edges such that the overal flow ϕ is a valid flow. This will be maximal (in this example) in all cases. Draw into the figure a cut that shows that ϕ is indeed maximal.

/5P



Aufgabe 2: Asymptotik (16P)

/3P (a) Finden Sie in der Liste der in der weissen Box angebenen Funktionen noch drei Paare von Funktionen f und g für die gilt $\Theta(f) = \Theta(g)$.

Find in the white boxed list of provided functions three more pairs of functions f and g such that $\Theta(f) = \Theta(g)$.

$\cancel{3n+3}$, $\log n$, $n \log n^2$, n^4, $n \log n$, n^2, $\cancel{\frac{n}{3}}$, $n^2 \log n$, $n \log^2 n$, $n!$
n^n, $\sum_{i=1}^{n} i \cdot \log n$, $\sum_{i=1}^{n} i^2$, $\sum_{i=1}^{n^2} i$

Beispiel / *Example*: $\Theta($ $n/3$ $) = \Theta($ $3n+3$ $)$

$\Theta($ $\Sigma_{i=1}^{n}\, i \cdot \log n$ $) = \Theta($ $n^2 \log n$ $)$

$\Theta($ n^4 $) = \Theta($ $\Sigma_{i=1}^{n^2}\, i$ $)$

$\Theta($ $n \log n$ $) = \Theta($ $n \log n^2$ $)$

(b) Gegeben Sei die folgende Rekursionsgleichung: *Consider the following recursion:* /6P

$$T(n) = \begin{cases} 2 + T(\frac{n}{3}), & n > 1 \\ 5 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (nicht rekursive), einfache Formel für $T(n)$ an und beweisen Sie diese mittels vollständiger Induktion. Gehen Sie davon aus, dass n eine Potenz von 3 ist.

Provide a closed (non-recursive), simple formula for $T(n)$ and prove it using mathematical induction. Assume that n is a power of 3.

$$\begin{aligned} T(3^k) &= 2 + T(3^{k-1}) \\ &= 2 + 2 + T(3^{k-2}) = \dots \\ &= \underbrace{2 + \dots + 2}_{k} + 5 \end{aligned}$$

Assumption: $T(n) = 5 + 2 \cdot \log_3 n$
Induktion:

1. Hypothesis $T(n) = f(n) := 2\log_3 n + 5$.

2. Base Case $T(1) = 5 = 5 + 2\log_3 1 = f(1)$.

3. Step $T(n) = f(n) \longrightarrow T(3 \cdot n) = f(3n)$ ($n = 3^k$ for some $k \in \mathbb{N}$):

$$\begin{aligned} T(3n) &= 2 + T(n) \\ &\overset{i.h.}{=} 2 + 2\log_3 n + 5 = 2\log_3 3n + 5 \\ &= f(3n). \end{aligned}$$

In den folgenden Aufgabenteilen wird jeweils angenommen, dass die Funktion g mit $g(n)$ aufgerufen wird. Geben Sie jeweils die asymptotische Anzahl von Aufrufen der Funktion $f()$ in Abhängigkeit von $n \in \mathbb{N}$ mit Θ-Notation möglichst knapp an. Die Funktion f ruft sich nicht selbst auf. Sie müssen Ihre Antworten nicht begründen.

In the following parts of this task we assume that the function g is called as $g(n)$. Provide the asymptotic number of calls of $f()$ depending on $n \in \mathbb{N}$ using Θ notation as tight as possible. The function f does not call itself. You do not have to justify your answers.

/1P (c)

```
void g(int n){
    f();
    if (n>0)
        g(n-1);
}
```

Anzahl Aufrufe von f / *Number of calls of f*

$\Theta(n)$

/2P (d)

```
void g(int n){
    if (n > 1){
        g(n-1);
        g(n-1);
    }
    f();
}
```

Anzahl Aufrufe von f / *Number of calls of f*

$\Theta(2^n)$

(e) Gegeben sei folgende Rekursionsgleichung: *Consider the following recursion equation:* /4P

$$T(n) = \begin{cases} 1 + T(\frac{n}{4}), & n > 1 \\ 2 & n = 1 \end{cases}$$

Schreiben Sie eine Funktion g, die bei Aufruf von $g(n)$ genau $T(n)$ Aufrufe von f erzeugt. Nehmen Sie an, dass $n = 4^k$ für ein $k \geq 0$.

Write a funciotn g that when called as $g(n)$ will produce $T(n)$ calls to f. Assume that $n = 4^k$ for some $k \geq 0$.

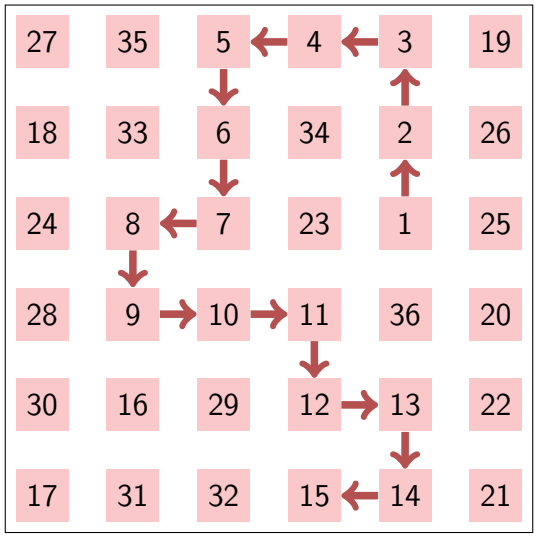
```
// pre: n = 4^k for some k >= 0
void g(int n){
    if (n == 1){
        f(); f();
    } else {
        f();
        g(n / 4);
    }
}
```

Aufgabe 3: Dynamic Programming (18P)

Maximale Länge einer Schlange. / *Maximum Length of a Snake.*

Gegeben sei ein zweidimensionales $n \times n$ Array $A = (A_{i,j})_{1 \leq i,j \leq n}$ aus paarweise unterschiedlichen natürlichen Zahlen $A_{i,j} \in \mathbb{N}$. Zwei Felder des Arrays heissen benachbart, wenn sie sich an einer Seite berühren, d.h. $A_{i,j}$ ist benachbart mit $A_{i,j-1}$, $A_{i-1,j}$, $A_{i,j+1}$, $A_{i+1,j}$ (sofern sie existieren). Elemente am Rand sind entsprechend mit weniger Elementen benachbart.

Consider an $n \times n$ matrix $A = (A_{i,j})_{1 \leq i,j \leq n}$, filled with pairwise different integer values $A_{i,j} \in \mathbb{N}$. Two elements of the array are neighbours when they touch at one of the horizontal or vertical sides, i.e. $A_{i,j}$ is a neighbour of $A_{i,j-1}$, $A_{i-1,j}$, $A_{i,j+1}$, $A_{i+1,j}$ (as far as they exist). Elements at the border of the array obviously have less neighbours.



Das Ziel ist es, die längste Schlange zu finden. Eine Schlange der Länge $k \in \mathbb{N}$ ist eine Folge $x_1, x_2, \ldots, x_k$ von Elementen des Arrays A so dass x_i und x_{i+1} benachbart sind und so dass $x_{i+1} = x_i + 1$ für alle $1 \leq i < k$.
In der Abbildung oben ist die längste Schlange eingezeichnet.

Geben Sie einen Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die Länge der längsten Schlange berechnet.

Goal is to find the longest snake. A snake with length $k \in \mathbb{N}$ is a sequence $x_1, x_2, \ldots, x_k$ of elements of the array A such that x_i und x_{i+1} are neighbours and such that $x_{i+1} = x_i + 1$ for each $1 \leq i < k$. The longest snake is depicted in the figure above.

Provide a dynamic programming algorithm for computing the length of the longest possible snake.

/8P

(a) (I)

Was ist die Bedeutung eines Tabelleneintrags, und welche Grösse hat die DP-Tabelle T?

What is the meaning of a table entry and what is the size of the DP-table T?

Tabellengrösse / *table size*: $n \times n$ `Table` L

Bedeutung Eintrag / *entry meaning*:
L_ij: length of the longest possible snake ending at the index (i, j).

(II)

Wie berechnet sich ein Eintrag der Tabelle aus den vorher berechneten Einträgen?

How can an entry be computed from the values of previously computed entries?

$$
\begin{aligned}
L_{i,j} = \max\{1,\\
&1 + L_{i+1,j} \texttt{if} A_{i+1,j} + 1 = A_{i,j}),\\
&1 + L_{i,j+1} \texttt{if} A_{i,j+1} + 1 = A_{i,j}),\\
&1 + L_{i-1,j} \texttt{if} A_{i-1,j} + 1 = A_{i,j}),\\
&1 + L_{i,j-1} \texttt{if} A_{i,j-1} + 1 = A_{i,j}),\\
&\}
\end{aligned}
$$

(we omit boundary conditions here for brevity)

(III)

Beschreiben Sie, in welcher Reihenfolge und wie die Einträge berechnet werden können.

Describe in which order and how the entries can be computed.

We start with $L = 0$ (for all entries). The entries are computed in any order, but non-computed entries (visible via $L_{ij} = 0$ need to be recursively computed in order to be able to compute the neighbouring values of a snake.

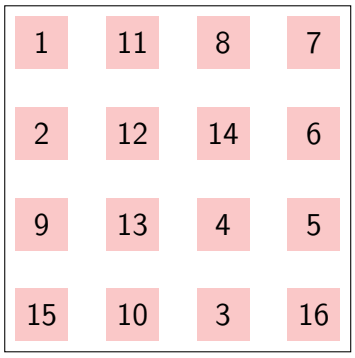
(IV) Wie kann der maximale Wert aus der Tabelle erhalten werden?

How can the maximal value be obtained from the DP table?

Largest value in L

/3P (b) Geben Sie die Tabelle für folgendes Beispiel an.

Provide the table for the follwoing example.




```
2  3  1  2
1  2  1  3
1  1  5  4
1  1  6  1
```

/4P (c) Zusätzlich zur Länge der längsten Schlange, wollen Sie nun die Schlange auch ausgeben. Beschreiben Sie, wie Sie das machen.

In addition to the maximal length of the snake, now you want to provide the longest snake itself. How do you do this?

```
Start from the beginning of the longest snake and go down through the
neighbours.
```

(d) Geben Sie die Laufzeiten und benötigten Speicherplatz für die Berechnung der maximalen Länge und der maximalen Schlange an.

Provide the run-times and the required extra memory space for the computation of the longest snake length and the longest snake.

/3P

The algorithm needs to visit each element of A (and L) only once. A and L have size n^2, so we have a running time and memory consumption of $\Theta(n^2)$ for all parts of the algorithm.

Aufgabe 4: Flussprobleme (17P)

In Anniks Klasse wird eine Telefonkette vereinbart für den Notfall. Dabei bekommt jedes Kind den Auftrag, anderen Kindern eine Nachricht telefonisch weiterzugeben, sobald es eine Nachricht erhält. Wer wen anruft wird im Vorhinein geklärt: jedes Kind erhält eine Liste von maximal 4 potentiellen Anrufzielen, dabei wird sichergestellt, dass Kinder sich nicht direkt gegenseitig anrufen: A ruft B an $\Rightarrow$ B ruft A nicht an. Beispiel:

In Anniks class at school a phone they organize a phone plan for urgent cases. Every child is instructed to pass on a message to other childern via phone as soon as it gets called. It is arranged up-front who calls who: every child gets a list of maximally 4 potential callees. It is made sure that children do not immediately call each other: A calls B $\Rightarrow$ B does not call A. Example:

Anrufer / *Caller*	**Empfänger** / *Receiver*
L	Tom, Mia, Lia
Tom	Alex, Annik
Mia	Alex
Lia	Mia, Hannes
Bert	Tom, Alex
Alex	Lia, Hannes, Annik
Hannes	Bert, Annik
Annik	Bert

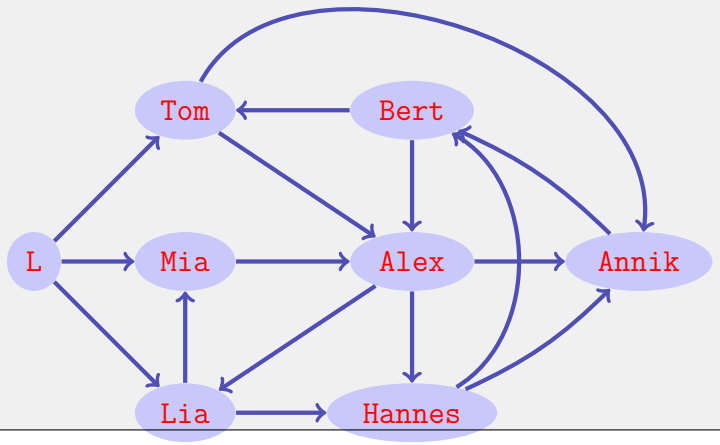
/4P (a) Sie wollen wissen, wie gut das Netzwerk für Annik ist: bekommt Sie die Nachricht von der Lehrperson L? Modellieren Sie das generelle Problem als Flussproblem und geben Sie das Flussnetzwerk an. Skizzieren Sie das Flussnetzwerk für obiges Beispielszenario (oder einen Teil davon).

You want to know how good the network is for Annik: does she get the message from the teacher L? Model the general problem as a flow problem. Describe the construction of the flow-graph. Sketch the flow network for (a part of) the example above.

1.children: nodes

2.telephone calls: edges

3.every edge has capacity 1



(b) Spezifizieren Sie nun einen möglichst effizienten Algorithmus zur Beantwortung folgender Frage: wie viele Telefonanrufe dürfen insgesamt maximal fehlschlagen, so dass Annik trotzdem garantiert die Nachricht der Lehrperson erhält? Tipp: kantendisjunkte Pfade.

Specify an efficient algorithm that can be used in order to answer the following question: how many calls may fail overal such that Annik is still guaranteed to get the message? Hint: edge-disjoint paths!

/4P

We need to know the number of edge-disjoint paths from L to Annik. Use the Ford-Fulkerson Algorithm in order to determine the maximum flow from s to t. The value of the maximum flow k determines the number $k-1$ of telephone calls that may go wrong because it also determines the min-cut of the graph.

(c) Geben Sie die Laufzeit Ihres Algorithmus (im schlechtesten Fall) in Abhängigkeit von der Anzahl Kinder mit kurzer Begründung an.

Provide the running time of your algorithm (in the worst case) as a function of the number of children n with short explanation.

/5P

Running time of the Edmonds-Karp algorithm is $O(V \cdot E^2)$, here this is $O(V^3)$
We have $O(E) = O(V) = O(n)$ and thus the running time is bounded by $O(n^3)$. However, a tighter estimation can be reached by considering the actual max flow that can happen, which is bounded by 4 on the last node. The running time of the Ford-Fulkerson Method is thus bounded by $O(E \cdot f^*) = O(n \cdot 4) = O(n)$.

(d) Bis hierher sind wir davon ausgegangen, dass einzelne Anrufe fehlschlagen. Nun möchten wir wissen, was passiert, wenn ein oder mehrere Kinder gar nicht erreichbar sind. Schlagen Sie einen Algorithmus vor, mit dem Sie nachprüfen können, wie viele Kinder ausfallen können und Annik trotzdem die Nachricht erhält.

Up to this point, we have assumed that single calls fail. Now we want to know what happens if one or more children are not available at all. Propose an algorithm that you can use in order to check how many children can get unavailable while Annik still gets the message.

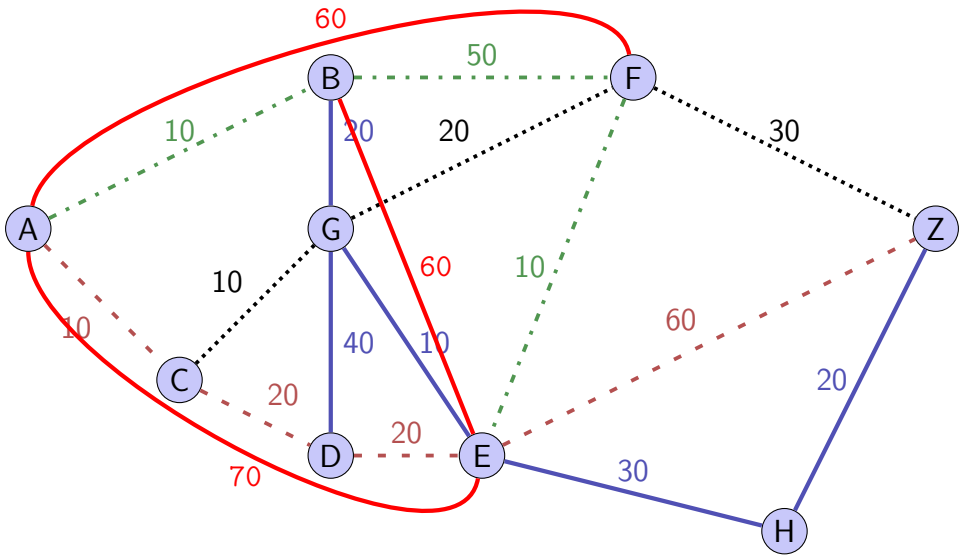
/4P

We are looking for the number of vertex-disjoint paths from L to A. The node for each child now becomes a connected pair of nodes i and o where i is connected to all incoming edges and o to all outgoing edges. There is a connection from i to o with capacity 1.

Aufgabe 5: Kürzeste Wege (10P)

Sie wollen von A nach Z verreisen und suchen die kostengünstigste Verbindung. Es gibt verschiedene Verkehrsmittel, die sie wählen können. Wir gehen davon aus, dass sich die Fahrkosten ergeben als die Summe der (strikt positiven) Kosten auf allen Teilstrecken.
In folgender Abbildung sehen Sie beispielhaft einen Graphen (V, E, w) der die Kosten pro Teilstrecke darstellt. Linien, welche ohne Umsteigen durchfahren werden können, sind farbig und mit verschiedenen Mustern markiert. So kann zum Beispiel der Pfad $A-B-F-E$ ohne Umsteigen durchfahren werden. Das ist für die ersten Aufgabenteile nicht wichtig.

You want to travel from A to Z and take the cheapest possible connection. There are different means of transportation that you can choose from. We assume that the travel cost are the sums of the (strictly positive) costs on all route sections.
In the following figure you see an exemplary graph (V, E, w) that depicts the costs per route sections. Moreover, the routes that can be travelled without changing means of transporation are marked with with different colors and patterns. For example, the route A - B - F - E can be taken without change. This is not important for the first parts of this task.



/3P (a) Welchen Algorithmus schlagen Sie zur möglichst effizienten Bestimmung des günstigsten Pfades vor?

Which algorithm do you propose in order to efficiently determine the cheapest path?

Dijkstra Algorithm

Was ist die asymptotische Laufzeit des Algorithmus in Abhängigkeit von der Anzahl Knoten $|V|$ und Anzahl Kanten $|E|$?

What is the asymptotic runtime of your algorithm as a function of the number of nodes $|V|$ and edges $|E|$?

$O((|E| + |V|) \log |V|) = O(|E| \log |V|)$

(b) Zeichnen Sie in obigem Graphen die transitive Hülle des (grün gestrichtelten) Pfades (Teilgraphen) $A-B-F-E$ ein und versehen Sie jede neue Kante mit den entsprechenden Fahrtkosten auf dieser Linie.

Draw in the graph above the transitive closure of the (dashed green) paths (sub graphs) $A-B-F-E$ and provide each new edge with the respective travelling costs on that line.

/1P

(c) Wir nehmen nun zum Graphen (V, E, w) die transitive Hülle jeder Linie hinzu (wie im vorigen Teil für eine Linie am Beispiel ausgeführt) und entfernen für jedes Paar Knoten alle Kanten bis auf die günstigste. Diesen neuen Graphen bezeichnen wir mit (V, E', w').
Sie wollen nun den günstigsten Pfad von A nach Z unter der Nebenbedingung finden, dass Sie nur bis zu $k > 0$, zum Beispiel $k = 2$, mal umsteigen müssen. Sie wollen auch detektieren, wenn ein solcher Pfad nicht existiert. Welchen Algorithmus schlagen Sie zur möglichst effizienten Bestimmung des günstigsten Pfades vor?

We now add to the graph (V, E, w) the transitive closure of each type of line (as exemplified in the previous part for just one line) and then remove all but the cheapest edge between each pair of vertices. We denote this new graph (V, E', w').
Now you want to find the cheapest path from A to Z given that you change transportation means only up to $k > 0$, for example $k = 2$ times. You also want to detect if such a path does not exist at all. Which algorithm do you propose to efficiently determine the cheapest path?

/4P

We can run the outer loop of the Bellman-Ford algorithm k times on (V', E', w').

(d) Geben Sie die asymptotische Laufzeit des Algorithmus in Abhängigkeit von den Parametern des neuen Graphen (V, E', w') und k an.

Provide the asymptotic runtime of the algorithm as a function of the parameters of the new graph (V, E', w') and k.

/2P

$O(k \cdot |E'|)$.

Aufgabe 6: Parallele Programmierung (17P)

/4P (a) Niklas hat sein sequentielles Programm parallelisiert und den sequentiellen Anteil seines neuen Programmes mit 0.2 spezifiziert. Er misst mit 3 Prozessoren eine Laufzeitverbesserung (Speedup) von 2.5. Mehr Details hat er leider nicht preisgegeben. Welches Gesetz (Amdahl oder Gustavson oder beide?) kann diese Laufzeitverbesserung erklären?

Nilas has parallelized his sequential program and specifies the sequential part of his new program with 0.2. Using 3 processors, he measures a runtime speedup of 2.5. More details he does not provide. Which law (Amdahl or Gustavson or both) can explain this runtime improvement?

Number processors $p = 3$, sequential part $\lambda = 0.2$ Speedup according to Amdahl

$$S_A \leq \frac{1}{1/5 + 4/5/3} = \frac{3}{3/5 + 4/5} = \frac{15}{7} < 2.5$$

Speedup according to Gustavson

$$S_G \leq p(1-\lambda) + \lambda = 3 \cdot 4/5 + 1/5 = 13/5 = 2.6 > 2.5$$

Only Gustavson can explain the speedup.

/3P (b) Erklären Sie kurz, warum in manchen Fällen das Gesetz von Gustavson angebracht ist und in anderen Fällen das Gesetz von Amdahl.

Shortly explain, why in some cases Gustavson's law is appropriate and in other cases the law of Amdahl.

Both laws are models of the speedup for parallelisation. Gustavson's law applies when the sequential part does not depend on the problem size (i.e. stays fixed when the problem size increases, true for many embarassingly parallel problems, such as pixel shading) while Amdahls law applies when the sequential part depends on the problem size (true for many divide-and-conquer algorithms).

Im nun folgenden Teil der Aufgabe werden Threads ausgeführt. Wir gehen jeweils davon aus, dass die Funktion `print` jeweils einen Buchstaben ausgibt (Thread-sicher).

In the following parts of the task, threads are executed. We assume for each part that the function `print` *outputs a character (threads-safe).*

Geben Sie jeweils alle möglichen Ausgaben der Programme darunter an (allenfalls getrennt durch Semikolon). Bestimmen Sie, ob das Programm terminiert. Wenn das Programm nicht terminiert, geben Sie alle Ausgaben immer so weit an, wie sie auftreten können.

Provide for each part all (theoretically) possible outputs (separated by semicolons) of the following programs below the program. Specify if the program terminates. If the program does not terminate, provide all possible outputs as far as they can occur.

(c) /2P

```
void print(char c); // output character c

void A(){
    print('A');
    print('B');
}

void B(){
    print('C');
    print('D');
}

int main(){
    std::thread t1(A);
    t1.join();
    std::thread t2(B);
    t2.join();
}
```

mögliche Ausgabe(n)/*possible output(s)*

ABCD

Das Programm terminiert/*the program terminates*

√ immer/*always* ◯ nie/*never* ◯ weder immer noch nie/*neither always nor never*

/2P (d)

```
void print(char c); // output character c

void A(){
    print('A');
    print('B');
}

void B(){
    print('C');
    std::thread t(A);
    print('D');
    t.join();
}

int main(){
    std::thread t(B);
    t.join();
}
```

mögliche Ausgabe(n)/*possible output(s)*

CABD; CADB; CDAB

Das Programm terminiert/*the program terminates*

√ immer/*always* ◯ nie/*never* ◯ weder immer noch nie/*neither always nor never*

(e) /2P

```
void print(char c); // output character c
std::mutex m; // global variable useable by all threads

void A(){
    m.lock();
    print('A');
    print('B');
    m.unlock();
}

void B(){
    print('C');
    std::thread t(A);
    print('D');
    t.join();
}

int main(){
    std::thread t(B);
    t.join();
}
```

mögliche Ausgabe(n)/*possible output(s)*

CABD; CADB; CDAB

Das Programm terminiert/*the program terminates*

√ immer/*always* ◯ nie/*never* ◯ weder immer noch nie/*neither always nor never*

/2P (f)

```
void print(char c); // output character c
std::mutex m; // global variable useable by all threads

void A(){
    m.lock();
    print('A');
    print('B');
    m.unlock();
}

void B(){
    m.lock();
    print('C');
    std::thread t(A);
    print('D');
    t.join();
    m.unlock();
}

int main(){
    std::thread t(B);
    t.join();
}
```

mögliche Ausgabe(n)/*possible output(s)*

```
CD;
```

Das Programm terminiert/*the program terminates*

◯ immer/*always* √ nie/*never* ◯ weder immer noch nie/*neither always nor never*

(g) /2P

```
void print(char c); // output character c
std::mutex m1; // global variable useable by all threads
std::mutex m2; // global variable useable by all threads

void A(){
    m2.lock();
    m1.lock();
    print('A');
    print('B');
    m1.unlock();
    m2.unlock();
}

void B(){
    m1.lock();
    print('C');
    std::thread t(A);
    m2.lock();
    print('D');
    m2.unlock();
    m1.unlock();
    t.join();
}

int main(){
    std::thread t(B);
    t.join();
}
```

mögliche Ausgabe(n)/*possible output(s)*

C; CDAB;

Das Programm terminiert/*the program terminates*

◯ immer/*always* ◯ nie/*never* √ weder immer noch nie/*neither always nor never*

/10P

Aufgabe 7: Parallele Programmierung (10P)

Ein zirkulärer (beschränkter) Puffer ist ein Objekt, in dem eine beschränkte Anzahl von Elementen gespeichert werden kann. Die Elemente werden nach dem FIFO Prinzip geschrieben (`put`) und gelesen (`get`). Nachfolgend ist ein solcher zirkulärer Puffer (nicht Thread-safe) implementiert. Der Puffer ist parameterisiert mit Elementtyp `T` und Puffergrösse `size`.
Passen Sie den nachfolgenden Code so an, dass der Puffer Thread-safe ist und so dass ein schreibender Thread jeweils wartet, bis Platz zum Schreiben da ist und ein lesender Thread jeweils wartet, bis Daten im Buffer vorhanden sind.

A circular (bounded) buffer is an object that can be used in order to store a limited number of elements. The elements are written (`put`) and read (`get`) according to the FIFO principle. In the following such a circular buffer is implemented (in a non thread-safe way). The buffer is parameterized with element type `T` and buffer size `size`.
Complement the following code such that the buffer is thread-safe and such that a writing thread waits until there is space to write into the circular buffer and a reading thread waits until data are available to read from the buffer.

```
#include <mutex> // for std::mutex and std::unique_lock
#include <condition_variable> // for std::condition_variable

template <typename T, int size>
class CircularBuffer{
  T buf[size];
  unsigned int in; unsigned int out;
    std::mutex m;
    using guard = std::unique_lock<std::mutex>;
    std::condition_variable cond;
public:
  CircularBuffer():in{0},out{0}{};

  bool empty(){
    return in == out;
  }
  bool full(){
    return (in + 1) % size == out;
  }
  void put(T x); // implementation on right hand side
  T get(); // implementation on right hand side
};
```

```
template <typename T, int size>
void CircularBuffer<T,size>::put(T x){
    guard g(m);
    cond.wait(g, [&]{return !full();});
  assert(!full());
  buf[in] = x;
  in = (in + 1) % size;
    cond.notify_all();
}

template <typename T, int size>
T CircularBuffer<T,size>::get(){
    guard g(m);
    cond.wait(g, [&]{return !empty();});
  assert(!empty());
  T x = buf[out];
  out = (out + 1) % size;
    cond.notify_all();
  return x;
}
```