Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prüfung **(Lösung)**

Datenstrukturen und Algorithmen (D-MATH RW)

Felix Friedrich, Pesho Ivanov, Departement Informatik

ETH Zürich, 5.8.2019.

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: 150 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: 150 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	17	16	17	14	18	118
Score:								

Verwenden Sie die Notation, Algorithmen und Datenstrukturen aus der Vorlesung. Falls Sie eine andere Herangehensweise wählen, erklären Sie Ihre Antworten in nachvollziehbarer Weise!

Use notation, algorithms and data structures from the course. If you use a different approach, explain your answers in a comprehensible way!

Aufgabe 1: Verschiedenes (20P)

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.

/2P (a) Das folgende Array ist nach dem ersten Aufteilungsschritt des Quicksort-Algorithmus (in-situ ausgeführt) entstanden. Markieren Sie alle Elemente des Arrays, welche dafür in Frage kommen, der Pivot des Aufteilungsschrittes gewesen zu sein.

The following array resulted from the first divide step of the Quicksort algorithm (executed inplace). Mark all elements of the array that could possibly have been the pivot of the divide step.

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

/2P (b) Geben Sie jeweils einen Algortihmus an, welcher das gegebene Problem auf einem **unsortierten Array im schlechtesten Fall** asymptotisch möglichst effizient löst. Wenn Ihnen kein Name zum Algorithmus bekannt ist, charakterisieren Sie den Algorithmus in 2 Worten.

Name an algorithm that solves the given problem on an **unsorted array in the worst case** *in the asymptotically most efficient way. If you do not know the name of an algorithm, characterize the algorithm in two words.*

Sortieren / Sort

Algorithmus *Algorithm*	Asymptotische Laufzeit *Asymptotic Running Time*
`Mergesort / Heapsort (NOT: Quicksort)`	$O(n \log n)$

Berechnung des Mittelwertes / Computation of the mean (average)

Algorithmus *Algorithm*	Asymptotische Laufzeit *Asymptotic Running Time*
`Sum up and divide`	$O(n)$

Berechnung des Median / Computation of the median

Algorithmus *Algorithm*	Asymptotische Laufzeit *Asymptotic Running Time*
Blums Algorithm	$O(n)$

Finde kleinstes Element / Find smallest element

Algorithmus *Algorithm*	Asymptotische Laufzeit *Asymptotic Running Time*
Elementwise comparison	$O(n)$

(c) Tragen Sie in folgendem Array die Zahlen 8, 12, 13 und 15 ein, so dass das Array einen Min-Heap darstellt.

Enter into the following array the numbers 8, 12, 13 and 15 such that the array constitutes a Min-Heap.

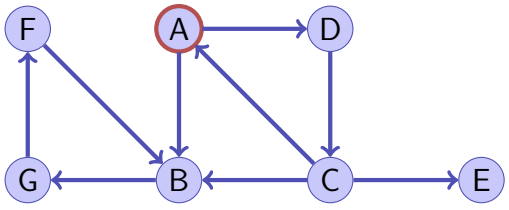
/2P

1	7	12	8	10	13	17	9	11	20	31	15	14
0	1	2	3	4	5	6	7	8	9	10	11	12

(d) Der folgende Graph wird, ausgehend vom Knoten A aus mit Breitensuche und mit Tiefensuche besucht. Wenn es mehrere Möglichkeiten für eine Besuchsreihenfolge der Nachbarn gibt, wird die alphabetische Ordnung genommen. Wie viele Knoten (inkl. A) besucht der Algorithmus **bis er einen Zyklus entdeckt**?

The following graph is visited with a breadth-first search and a depth-first search algorithm starting at node A. If there are several possibilities for a visiting order of the neighbours, the alphabetical order is chosen. How many nodes (including A) are visited **until the algorithm detects a cycle***?*

/2P



Anzahl besuchter Knoten Tiefensuche/*Number of visited nodes Depth First Search*:

4 (A,B,G,F)

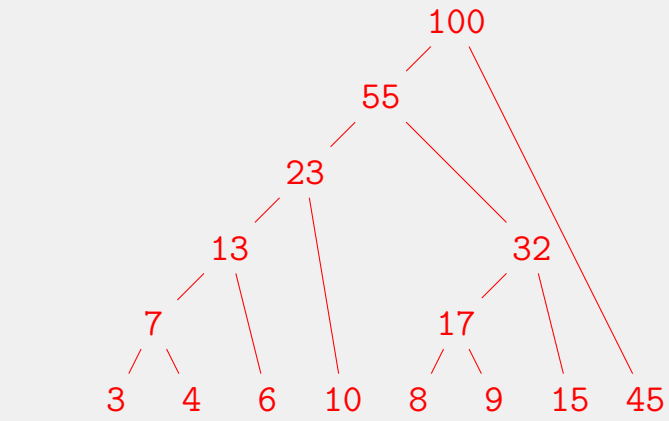
Anzahl besuchter Knoten Breitensuche/*Number of visited nodes Breadth First Search*:

6
(A,B,D,G,C,F)

/2P (e) Gegeben sind acht Buchstaben mit Häufigkeit (Anzahl Zugriffe) wie folgt. Erstellen Sie mit Hilfe des Huffman-Algorithmus einen optimalen Codierungsbaum. Tragen Sie den resultierenden Code in der Tabelle ein.

Eight characters (keys) with occurences (number of accesses) are given as follows. Using the Huffman algorithm construct an optimal code tree. Enter the corresponding code into the table.

char	a	b	c	d	e	f	g	h
freq	3	4	6	10	8	9	15	45
Code	00000	00001	0001	001	0100	0101	011	1



(meaning of 0 and 1 can be exchanged)

/2P (f) Fügen Sie die folgenden Schlüssel in der angegebenen Reihenfolge mit Double-Hashing in die Hashtabelle ein. Die verwendeten Hash-Funktionen $h(k)$ und $h'(k)$ sind nachfolgend angegeben (beim Double-Hashing wird addiert).
Tipp: damit Sie sich nicht verrechnen, notieren Sie die Werte von h und h' jeweils bei den Schlüsseln.

Enter the following keys in the given order into the hash-table using Double-Hashing. The used hash functions $h(k)$ and $h'(k)$ are stated below (double hashing is assumed to use summation).
Hint: in order to avoid miscalculations, write down the values of h and h' for each key.

Hashfunktionen / *hash functions* : $h(k) = k \bmod 13$, $h'(k) = 1 + (k \bmod 11)$

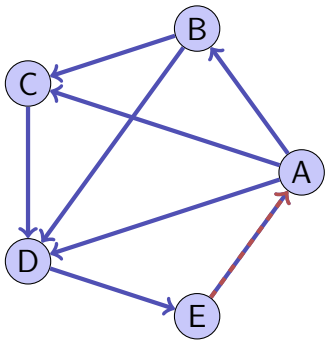
Schlüssel / *keys* : 13, 26, 39, 1, 14, 27, 40

13	1			40	26		39	27	14			
0	1	2	3	4	5	6	7	8	9	10	11	12

(g) Streichen Sie in folgendem Graphen eine kleinstmögliche Menge von Kanten, so dass der verbleibende Graph eine topologische Sortierung hat.

In the following graph, cross out the smallest possible set of edges such that the remaining graph can be topologically sorted.

/2P



(DE also possible)

(h) Ein ungerichteter Graph G hat n Knoten. Seine Adjazenzmatrix ist durch eine $n \times n$ Matrix gegeben. Alle Einträge auf der Diagonale seien 0, alle anderen 1. Welche der folgenden Antworten stimmt?

An undirected graph G has n vertices. The adjacency matrix is given by an $n \times n$ matrix. All diagonal entries are 0 and all other entries are 1. Which of the following is true?

/1P

◯ G hat keinen minimalen Spannbaum /
G has no minimum spanning tree

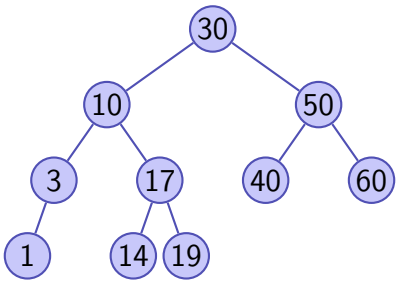
◯ G hat einen einzigen minimalen Spannbaum mit Kosten 1 /
G has a unique minimum spanning tree with costs 1

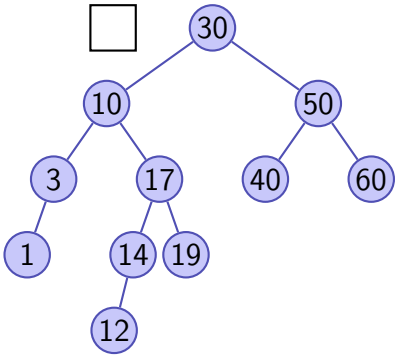
√ G hat mehrere verschiedene minimale Spannbäume, jeweils mit Kosten $n-1$ /
G has multiple distinct minimum spanning trees, each with costs $n-1$

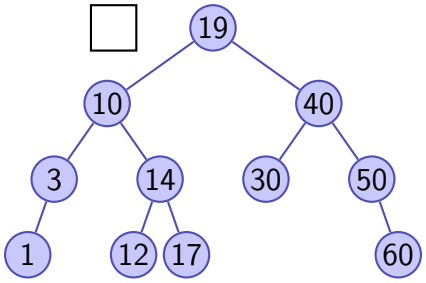
◯ G hat mehrere verschiedene minimale Spannbäume mit verschiedenen Kosten /
G has multiple distinct minimum spanning trees with different costs.

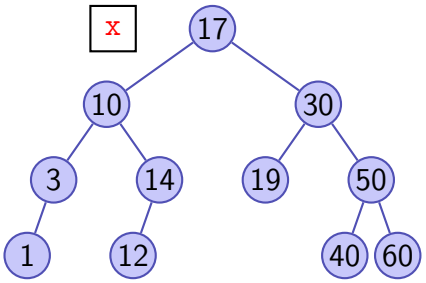
/2P (i) Fügen Sie in folgendem AVL Baum den Schlüssel 12 ein und rebalancieren Sie (wie in der Vorlesung gezeigt). Wie sieht der AVL Baum nach der in der Vorlesung gezeigten Operation aus? Kreuzen Sie die richtige Antwort an.

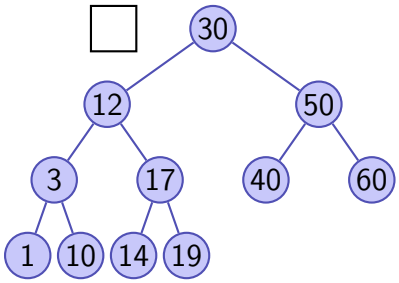
In the following AVL tree, insert key 12 and rebalance (as shown in class). What does the AVL tree look like after the operation that has been shown in class? Mark the correct answer.

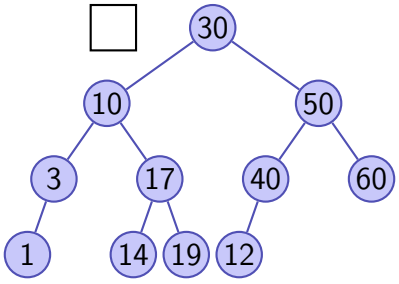


☐


☐


☒ x


☐


☐


/3P

(j) Nennen Sie zu jedem der folgenden Operationen auf einer Menge von ganzen Zahlen eine Datenstruktur, die Sie zur asymptotisch möglichst effizienten Implementation einsetzen würden. Die genannten Operationen werden sehr oft ausgeführt. Sofern sich die Datenstrukturen in der asymptotischen Effizienz nicht unterscheiden, wählen Sie die einfachere.

To the following operations on a set of integers, name a data structure that you would use for an implementation, asymptotically as efficient as possible. The specified operations are executed very often. If several data structures do not differ in their asymptotic efficiency, choose the simplest possible.

1. Am Anfang einfügen und nach Einfügereihenfolge durchlaufen.

   Insert at the beginning and traverse in the order of insertion

   ```
   Linked List
   ```

2. Einfügen, Löschen und Suchen von Elementen.

   Insertion of, removal of and search for elements.

   ```
   Hash-table (or balanced tree)
   ```

3. Einfügen, Löschen, Finde zu gegebenem Element das nächst grössere.

   Insertion of, removal of elements, search the next larger element for a given element.

   ```
   Balanced tree (AVL Tree)
   ```

4. Einfügen, kleinstes Element extrahieren.

   Insertion of elements, extraction of the smallest element.

   ```
   Min-Heap
   ```

5. Mengen vereinen und prüfen, ob Paare in derselben Menge enthalten sind.

   Union of sets and check if pairs of elements are contained in the same set.

   ```
   Union-Find data structure
   ```

6. Einfügen, Löschen, Median berechnen.

   Insertion, removal, compute the median.

   ```
   Balanced Tree (AVL) or two Heaps (Min-/Max)
   ```

Aufgabe 2: Asymptotik (16P)

/3P (a) Geben Sie für die untenstehenden Funktionen eine Reihenfolge an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \in \mathcal{O}(g)$.
Beispiel: die drei Funktionen n^3, n^5 und n^7 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^5)$ und $n^5 \in \mathcal{O}(n^7)$.

Provide an order for the following functions such that the following holds: If a function f is left of a function g then it holds that $f \in \mathcal{O}(g)$.
Example: the functions n^3, n^5 and n^7 are already in the respective order because $n^3 \in \mathcal{O}(n^5)$ and $n^5 \in \mathcal{O}(n^7)$.

$$\log \sqrt{n} \ , \ n^{2n} \ , \ \sqrt{\log n} \ , \ n! \ , \ \sum_{i=1}^{n} i \ , \ n \log n \ , \ 10^{10}$$

10^{10}	$\sqrt{\log n}$	$\log \sqrt{n}$	$n \log n$	$\sum_{i=1}^{n} i$	$n!$	n^{2n}

(b) Gegeben sei die folgende Rekursionsgleichung:

Consider the following recursion equation:

/6P

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

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 2 ist.
Hinweis:
Für $q \neq 1$ gilt $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

Specify a closed (non-recursive), simple formula for $T(n)$ and prove it using mathematical induction. Assume that n is a power of 2.
Hint:
For $q \neq 1$ it holds that $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + 2^k/2 + 1 \\ &= 2(2(T(2^{k-2}) + 2^{k-1}/2 + 1) + 2^k/2 + 1 = \dots \\ &= 2^k T(2^{k-k}) + \underbrace{2^k/2 + \dots + 2^k/2}_{k} + 1 + 2 + \dots + 2^{k-1} \\ &= 3n + \frac{n}{2}\log_2 n + n - 1 \end{aligned}$$

`Assumption:` $T(n) = 4n + \frac{n}{2}\log_2 n - 1$
`Induktion:`

`1.Hypothesis` $T(n) = f(n) := 4n + \frac{n}{2}\log_2 n - 1$

`2.Base Case` $T(1) = 3 = f(1) = 4 - 1.$

`3.Step` $T(n) = f(n) \longrightarrow T(2 \cdot n) = f(2n)$ `(`$n = 2^k$ `for some` $k \in \mathbb{N}$`):`

$$\begin{aligned} T(2n) &= 2T(n) + n + 1 \\ &\overset{i.h.}{=} 2(4n + \frac{n}{2}\log_2 n - 1) + n + 1 \\ &= 8n + n\log_2 n - 2 + n + 1 \\ &= 8n + n\log_2 n + n\log_2 2 - 1 \\ &= 8n + n\log_2 2n - 1 \\ &= f(2n). \end{aligned}$$

In den folgenden Aufgabenteilen wird jeweils angenommen, dass die Funktion g mit $g(n)$ aufgerufen wird ($n \geq 0$). 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 $g(n)$ ($n \geq 0$). Specify the asymptotic number of calls of $f()$ depending on $n \in \mathbb{N}$ using Θ notation as succinct 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>1)
        g(n/2);
        g(n/2);
}
```

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

$\Theta(n)$

/3P (d)

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

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

$\Theta(2^n)$

/3P (e) Gegeben sei folgende Rekursionsgleichung:

Consider the following recursion equation:

$$T(n) = \begin{cases} 2T(n/2) + n, & n > 1 \\ 0 & n \leq 1 \end{cases}$$

Schreiben Sie eine Funktion g, die bei Aufruf von $g(n)$ genau $T(n)$ Aufrufe von f erzeugt. Nehmen Sie an, dass $n = 2^k$ für ein $k \geq 0$.

Write a function g that when called as $g(n)$ will produce $T(n)$ calls to f. Assume that $n = 2^k$ for some $k \geq 0$.

```
// pre: n = 2^k for some k >= 0
void g(int n){
        if (n > 1){
        g(n/2); g(n/2);
        while(n-- > 0){
        f();
        }
    }
}
```

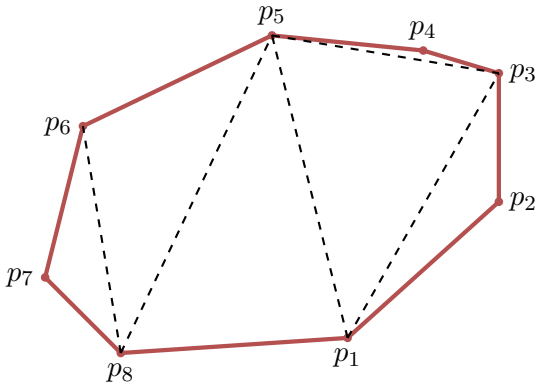
Seite bewusst freigelassen / *Page left free intentionally*

Aufgabe 3: Dynamic Programming (17P)

Triangulierung eines Polygons mit minimalen Kosten. / *Minimum Cost Polygon Triangulation.*

Gegeben sei ein Array mit $n \geq 3$ Punkten $p_i = (x_i, y_i)_{1 \leq i \leq n}$ in der Ebene. Die Punkte definieren in der gegebenen Reihenfolge ein konvexes Polygon: für alle Paare von aufeinander folgenden Punkten (p_i, p_j) mit $j = i+1$ oder $i = n, j = 0$ liegen alle anderen Punkte auf der Halbebene, die durch die Gerade durch p_i und p_j definiert ist. Wir nehmen an, dass keine 3 Punkte auf derselben Geraden liegen.

Consider an array with $n \geq 3$ points $p_i = (x_i, y_i)_{1 \leq i \leq n}$ in the plane. The points form a convex polygon in the order they are given: for all pairs of consequitive points (p_i, p_j), such that $i + 1 = j$ or $i = n, j = 0$, all other points lie in the same half-plane defined by p_i and p_j. Assume that no three points lie on a line.



Das Ziel ist die Berechnung der minimalen Kosten einer Triangulierung des Polygons. Eine Triangulierung des Polygons überdeckt dieses, wie in der Abbildung beispielhaft gezeigt, mit nicht-überlappenden Dreiecken. Jedes Dreieck besteht aus Ecken aus dem gegebenen Array. Die Kosten einer Triangulierung ist die Summe der Kosten aller Dreiecke.

The goal is to compute the minimum cost of a triangulation of the given polygon. A triangulation of a polygon is fully covering it with non-overlapping triangles, each of which has vertices from the given array, as depicted in the figure above. The cost of a triangulation is the sum of the costs of its triangles.

Geben Sie einen Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die minimalen Kosten einer Triangulierung eines gegebenen Polygons berechnet. Sie können davon ausgehen, dass die nicht-negative Funktion $C(a, b, c)$ zur Berechnung der Kosten des Dreickes $a - b - c$ bereits implementiert ist.

Specify a dynamic programming algorithm for computing the minimum cost of a triangulation of the given polygon. You can assume that the non-negative function $C(a, b, c)$ is already implemented and computes the cost of the triangle formed by the three points a, b and c.

Tipp: Nehmen Sie einen Tabellen-Eintrag für jedes Punkte-Paar.

Hint: You may consider a table entry for each pair of vertices.

/10P

(a) (I)

Was bedeutet ein Tabelleneintrag und wie viele Einträge der DP-Tabelle werden benötigt?

What is the meaning of a table entry and how many entries are needed from the DP-table T?

Tabellengrösse / *table size*:
$n \times (n-1)/2$ entries: for each pair of points $(p_i, p_j)_{i<j}$.

Bedeutung Eintrag / *entry meaning*:
For $1 \leq i < j \leq n$: T_{ij} is the minimum cost of a triangulation of the polygon formed by the points $p_i, p_{i+1}, \ldots, p_j$.

(II)

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

How can the entries be computed based on previously computed entries?

$$T_{i,j} = \begin{cases} \min_{k:i<k<j} \{T_{i,k} + T_{k,j} + C(p_i, p_k, p_j)\} & \text{if } i+1 < j, \\ 0 & \text{otherwise} \end{cases}$$

(III)

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

Describe in which order the entries can be computed.

We will iterate all polygons with vertices from i to j ($i < j$) from the smallest (in number of vertices) to the largest.
We start with the smallest polygons, namely triangles: each triangle has its first and last vertex two incides apart ($i+2=j$). One possible ordering is $(1,3),(2,4),\ldots,(n-2,n)$ but any order of the triangles works. Then 4-vertex polygons ($i+3=j$), then 5-vertex polygons and so on.

(IV)

Wie kann der minimale Wert der Triangulierungskosten aus der Tabelle erhalten werden?

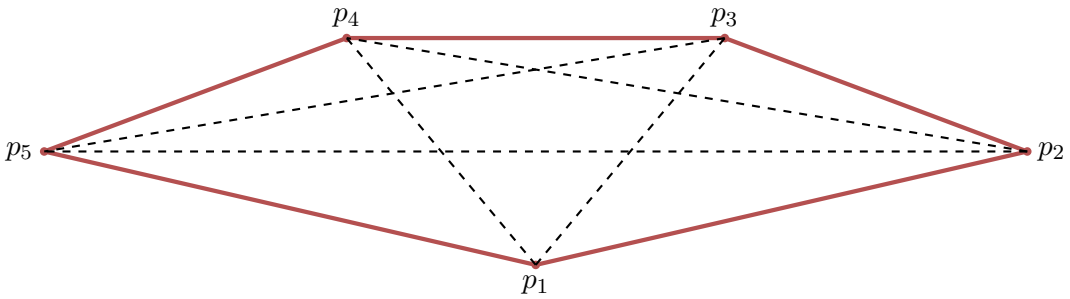
How can the minimal triangulation cost be obtained from the DP table?

The finally computed $T(1,n)$ is the answer.

/3P (b) Geben Sie die DP-Tabelle für folgendes Beispiel an. Die Kosten der Dreiecke sind angegeben. Welche Dreiecke sind Teil einer optimalen Triangulierung?

Calculate the DP-table for the following example, assuming the given triangle costs. Which triangles will be part of an optimal triangulation?

$C(1,2,3) = 3$ $C(1,3,4) = 3$ $C(2,3,4) = 3$ $C(3,4,5) = 3$
$C(1,2,4) = 4$ $C(1,3,5) = 4$ $C(2,3,5) = 5$
$C(1,2,5) = 5$ $C(1,4,5) = 3$ $C(2,4,5) = 5$



	1	2	3	4	5
1		0	3	6	9
2			0	3	6
3				0	3
4					0
5					

The only optimal triangulation is defined by the triangles $\{(p_1, p_2, p_3), (p_1, p_3, p_4), (p_1, p_4, p_5)\}$.

Explanations:

$$T(1,4) = \min(T(1,3) + T(3,4) + C(1,3,4), T(1,2) + T(2,4) + C(1,2,4))$$
$$= \min(3 + 0 + 0, 0 + 3 + 4) = 3$$
$$T(2,5) = \min(T(2,3) + T(3,5) + C(2,3,5), T(2,4) + T(4,5) + C(2,4,5))$$
$$= \min(0 + 3 + 3, 3 + 0 + 5) = 6$$
$$T(1,5) = \min(T(1,2) + T(2,5) + C(1,2,5), T(1,3) + T(3,5) + C(1,3,5), T(1,4) + T(4,5) + C(1,4,5))$$
$$= \min(0 + 6 + 5, 3 + 3 + 4, 6 + 0 + 3) = 9$$

(c) Zusätzlich zu den optimalen Kosten einer Triangulierung wollen Sie nun die beste Triangulierung selbst angeben. Wie könnten Sie die Triangulierung repräsentieren und wie berechnen Sie diese?

In addition to the optimal cost of a triangulation, now you want to determine the triangulation itself. How would you represent a triangulation and how do you compute it?

/2P

The triangulation can be represented different ways. One is as a set of triangles. Another is as a set of edges dividing the polygon to triangles. Whenever you maximize a table entry, also remember which third vertex k you have used. After computing the table, use this information to recover the minimum cost triangulation. Another way to recover the best step after computing the table is to recompute the step costs using $C(a, b, c)$.

(d) Geben Sie die asymptotischen Laufzeiten für den schlechtesten Fall und den benötigten zusätzlichen Speicherplatz (einschliesslich der Tabelle) für Ihren Algorithmus zur Berechnung der minimalen Triangulierungskosten an. Wie ändern sich Laufzeit und Speicherplatzbedarf, wenn man die minimale Triangulierung auch angeben will?

Specify the worst case asymptotic runtimes and the required extra memory space (incl. for the table) for your algorithm used to compute the minimum triangulation cost. How will those change if the determination of such a minimal trianguation is also needed?

/2P

The presented algorithm for computing the minimum cost requires $\Theta(n^2)$ time and memory. The memory can be cut down to $\Theta(n)$ if the solution is further optimized to preserve only the previous diagonal.
Reporting a minimum triangulation along with its cost requires $\Theta(n^2)$ time and memory.

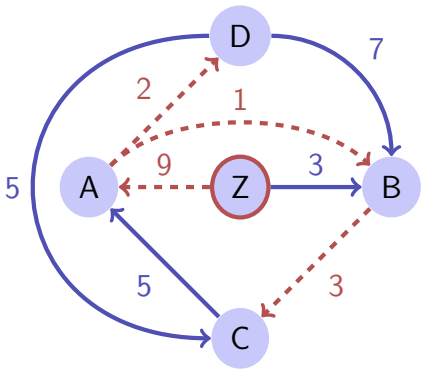
Aufgabe 4: Kürzeste Wege (16P)

Sie gehen **von einer Stadt Z** aus auf die Reise. Sie können eine Stadt mehrfach besuchen. Jedesmal, wenn Sie zur nächsten Stadt fahren, zahlen Sie entweder ein Ticket (Sie haben immer genug Geld), oder Sie fahren per Anhalter umsonst und bekommen, weil Sie auf der Fahrt so schöne Musik machen, sogar noch etwas Trinkgeld vom Fahrer. Wegen des Trinkgeldes können Sie sogar Geld machen! Allerdings können Sie nicht von einer Stadt wegfahren und zu der Stadt mit mehr Geld zurückkehren.

You start a journey **from town Z** *and you are going to change towns at each step, possibly arriving more than once to the same town. Every time you change town, you either pay for a ticket (you always have enough money), or hitchhike for free and receive a tip from the driver for the nice music you play during the ride. Because of these tips, it is even possible to make some money! Nevertheless, it is not possible to arrive for a second time to the same town and have more money than when you were there for a first time.*

In der folgenden Abbildung sehen Sie beispielhaft ein Netzwerk der Städte, welches Kosten / Gewinn pro Verbindung anzeigt. Gestrichelt rote Kanten stehen für eingenommenes Trinkgeld, während blaue durchgezogene Kanten für Ticketkosten stehen.

The following figure contains an example network of cities with cost / gain for the connections between them. Dashed red egdes denote tips and solid blue edges denote ticket payments.



/2P (a) Die Aufgabe ist, eine beste Route, also eine Route mit maximalem Gewinn zu finden. Formulieren Sie die Aufgabe als ein Kürzeste-Wege Problem. Dafür müssen Sie den Graphen und alle Nebenbedingungen definieren.

The task is to find a best trip, i.e. a trip with a maximum total gain. Formulate the task of planning the best trip as a shortest path problem. For this, you need to define the graph and all constraints.

Find a shortest path from Z to any other vertex in the graph $G(V, E, c)$, V=towns, E=transitions, c=$\{-$tips, $+$ticket costs$\}$) which does not have negative cycles.

(b) Nun wollen Sie eine Reise mit **genau L Stadtwechseln** machen. Kann der Algortihmus von Bellman-Ford direkt auf obiges Problem angewendet werden. Warum/warum nicht? Modifizieren Sie nötigenfalls den ursprünglichen Algorithmus und schätzen Sie die asymptotische Laufzeit im schlechtesten Fall ab (abhängig von der Anzahl Städte N und Anzahl Verbindungen M).

Now you want to make a trip with **exactly L transitions** *between towns. Can Bellman-Ford's algorithm be directly applied. Why/Why not? Modify the original algorithm if necessary and estimate the worst case asymptotic runtime (in terms of the number of towns N and the number of connections M).*

/4P

There is no problem running BF on G from the perspective of edge costs -- there are negative edges but no negative cycles (which is sufficient for BF).
The problem is with finding a shortest path with **exactly** L edges. If you run the outer loop of the Bellman-Ford algorithm L times on G, you can calculate shortest paths with **not more than** L edges (but possibly less) from Z to any other vertex. In order not to account for paths shorter than L, you can use $L+1$ arrays (from 0 to L incl.) instead of only one: the i-th array corresponding to paths with length exactly i.
Runtime complexity: same as the usual BF, $O(L \cdot M)$.

(c) Wenden Sie Ihre Variante des Bellman-Ford Algorithmus auf den Beispielgraphen an und geben Sie eine $(L+1) \times N$ Tabelle (für $L = 5$) mit den Kosten der kürzesten Pfade der Längen $0, 1, \ldots, L$ zu jedem der N Knoten an.

Apply your variant of Bellman-Ford's algorithm to the example graph from above and draw an $(L+1) \times N$ table (for $L = 5$) with the costs of the shortest paths of length $0, 1, \ldots, L$ to each of the N vertices.

/3P

	Z	A	B	C	D
0	0	∞	∞	∞	∞
1	∞	-9	3	∞	∞
2	∞	∞	-10	0	-11
3	∞	5	-4	-13	∞
4	∞	-8	4	-7	3
5	∞	-2	-9	1	-10

Shortest path cost (gain): $cost = -10 / gain = 10$

/4P (d) Angenommen, der Graph hat keine negativen Kanten. Erklären Sie, wie Sie einen neuen Graphen konstruieren würden, so dass der Dijkstra-Algorthmus benutzt werden kann (ohne den Algorithmus zu verändern), um einen kürzesten Pfad mit **genau** L Kanten zu finden. Was ist die asymptotische Anzahl Knoten und Kanten in diesem Graph in Abhängigkeit von der Anzahl Städte N und Anzahl Verbindungen M?

*Assume a graph without negative edges. Explain how to construct a new graph in which Dijkstra's algorithm can be run (without modifying the algorithm) in order to find the shortest path with length **exactly** L. What is the asymptotic number of vertices and edges in this new graph in terms of the number of towns N and the number of transitions M?*

Dijkstra will also not account for having exactly L edges in the path. A possible graph extension consists of two steps:

1) Copy the nodes $L+1$ times to produce a $L+1$-layered graph (being in a node on layer $i (0 \leq i \leq L)$ means you have made i transitions). Every transition should be from layer i to layer $i + 1$. A node this graph will be denoted with the pair $(town, layer)$ and the transitions will be copied L times between adjacent layers.

2) Use Dijkstra to find a shortest path from $(Z, 1)$ to (v, L) for any v.

The number of vertices in the constructed graph is $O(NL)$ and the number of edges is $O(ML)$.

/3P (e) Erklären Sie, wie man einen Graphen konstruieren kann, so dass man den Dijkstra Algorithmus (ungeändert) auf das oben beschriebene Problem mit Städten anwenden kann. Hinweis: achten Sie auf negative Kanten.

Explain how to construct such a graph on which you can run Dijkstra's algorithm (without modifying the algorithm) so that you solve the given task with the towns. Hint: Mind the negative edges.

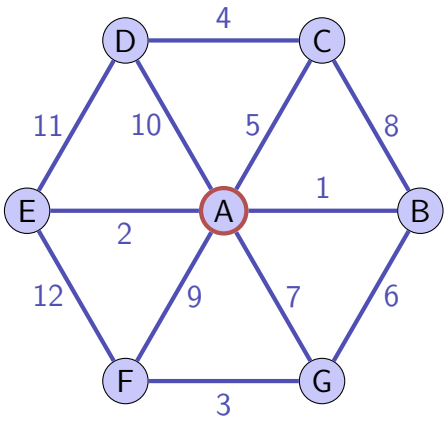
Dijkstra cannot directly deal with negative edges. Add the same positive constant C to all edges in order the make all of them non-negative (then subtract $C \cdot L$ from the cost of the shortest path). Choose C such that $C > -\min edgeweight$.

Seite bewusst freigelassen / *Page left free intentionally*

Aufgabe 5: Minimale Spannbäume (17P)

/2P (a) **Geben Sie zu folgendem Graphen die Kanten eines minimalen Spannbaumes an** in der Reihenfolge, in der sie ausgewählt würden durch die Algorithmen von Kruskal und Prim-Dijkstra. Gehen Sie davon aus, dass Prim-Dijkstra im **Knoten A** startet. Bezeichnen Sie die Kanten mit ihren Gewichten.

Consider the following graph. **Fill in the edges of a minimum spanning tree** *in the order that they would be selected by the algorithms of Kruskal and Prim-Dijkstra. Assume that Prim-Dijkstra starts with* **node A**. *Specify the edges using their weights.*



Algorithmus / *Algorithm*	Kanten / *edges*
Kruskal	1 2 3 4 5 6
Prim-Dijkstra	1 2 5 4 6 3

Nun werden Sie einen neuen **Algorithmus von Borůvka** (1926) kennenlernen. Bei diesem werden für die Berechnung eines minimalen Spannbaumes M ebenfalls getrennte (Zusammenhangs-)Komponenten verbunden. Borůvka wählt die Komponenten aber in einer anderen Weise. Der Algorithmus startet mit $M = \emptyset$ und mit n Komponenten: jede Komponente besteht aus einem einzigen Knoten. In jeder Runde des Algorithmus wird zu jeder Komponente eine kürzeste Kante zu einer anderen Komponente gewählt. Die Vereinigung dieser Kanten wird zu M hinzugenommen und durch die Kanten verbundene Komponenten werden vereinigt. Der Algorithmus terminiert, wenn eine einzige Zusammenhangskomponente übrig ist.

Now you will be presented a new algorithm for finding the minimum spanning tree: **Borůvka's algorithm** *(1926). It also merges disconnected components in order to compute a minimum spanning tree M. But it chooses the components in a different fashion. The algorithm starts with $M = \emptyset$ and with n components: each component consists of a single node. At every round of the algorithm, for each component a shortest edge to another component is selected. The union of these edges is then added to M and components that are connected via the selected edges are merged. The algorithm stops when a single connected component remains.*

(b) Geben Sie die gewählten Kanten und entstehende Zusammenhangskomponenten (in der Form {ABC}) des Algorithmus von Borůvka für die ersten zwei Runden an.

Fill in the selected edges and formed connected components (in the form {A,B,C}) of Borůvka's algorithm for the first two rounds.

/4P

Runde / *Round*	Kanten / *Edges*	Komponenten / *Components*
0.	–	{A},{B},{C},{D},{E},{F},{G}
1.	1 4 2 3	{A,B,E}, {C,D}, {F,G}
2.	5 6	{A,B,E,C,D,F,G}

(c) Angenommen alle Kanten haben verschiedenes Gewicht. Ist es möglich, dass es während einer Runde des Algortihmus von Borůvka drei Komponenten A, B und C gibt, so dass die leichteste Kante jeweils von A nach B, von B nach C und von C nach A gewählt wird? Wenn ja, geben Sie ein Beispiel an. Wenn nein, erklären Sie, warum nicht.

Assume all edges have different weights. Is it possible that during one round of Borůvka's algorithm there may be three connected components A, B and C such that the shortest edge from A is to B, from B is to C, and from C is to A. If yes, provide an example; if not, explain why not.

/3P

Such case is not possible.
Let e_{min} be the minimum edge out of the three edges forming the cycle between A, B and C. Lets call the components it connects X and Y. Because the edges are undirected, e_{min} is a candidate minimal edge for both X and Y. As all edges have different weights, any other outgoing edge from X and Y would be heavier than e_{min} (which contradicts the assumption that there are different edges outgoing from all three components).

(d) Die Algorithmen lassen sich mit Hilfe der Auswahl- und Verwerfregel aus der Vorlesung beweisen.

Auswahlregel: Wähle einen Schnitt, den keine gewählte Kante kreuzt. Unter allen unentschiedenen Kanten, welche den Schnitt kreuzen, wähle diejenige mit minimalem Gewicht.

The algorithms can be proven by using the Selection and the Rejection rule from class.

Selection rule: *choose a cut that is not crossed by a selected edge. Of all undecided edges that cross the cut, select the one with minimal weight.*

/4P

Verwerfregel: Wähle einen Kreis ohne verworfene Kanten. Unter allen unentschiedenen Kanten im Kreis verwerfe die mit maximalem Gewicht.

Rejection rule: *choose a cycle without rejected edges. Of all undecided edges of the cycle, reject those with maximal weight.*

Beschreiben Sie nun in maximal vier kurzen Sätzen, wie der Borůvka-Algorithmus obige Auswahl- und Verwerfregeln respektiert.

Describe in maximally four short sentences, how the Borůvka-Algorithm respects the selection- and rejection rules from above.

Borůvka chooses in each round the cheapest edge incident to a tree to a different tree. Borůvka rejects in each round the undecided edges connected to a chosen edge. By definition of the algorithm, the edges would have greater or same edge weight and would form a cycle in the trees.

/4P (e) Beschreiben Sie kurz eine mögliche Implementierung des Algorithmus von Borůvka. Geben Sie für den schlechtesten Fall die asymptotische Anzahl Runden und asymptotische Gesamtlaufzeit des Algorithmus (als eine Funktion der Anzahl Knoten $|V|$ und Kanten $|E|$) mit kurzer Begründung an.

Briefly describe a possible implementation and argument about the worst case asymptotic number of rounds and the runtime of the algortihm of Borůvka (as a function of the number nodes $|V|$ and edges $|E|$).

Using an adjacency list for each node, each round costs $|E|$ to choose the shortest edge outgoing from each component. Using an array of size $|V|$ with the component number for each vertex (or a list of nodes for each component), merging all components in one round costs $|V|$ (which is dominated by $|E|$ for that round). Because in each round each component gets connected to another component, the number of components remaining is at last halved. Therefore we have $O(\log |V|)$ rounds and overal costs of $O(|E| \cdot \log |V|)$.

Aufgabe 6: Parallele Programmierung (14P)

/4P

(a) Hannes hat ein Programm geschrieben und es ist leider nicht schnell genug. Hannes weiss, dass das Programm einen sequentiellen Anteil von 40% aufweist. Er hat zwei Optionen:

1. Er kauft einen Rechner A mit 4 Prozessorkernen und halbiert den sequentiellen Anteil.
2. Er kauft einen Rechner B mit 6 Prozessoren. Er weiss, dass B aufgrund seiner höheren Taktfrequenz generell eine bessere Performanz aufweist. Die Laufzeit verkürzt sich um weitere 10%.

Beraten Sie Hannes: welche Option verspricht eine bessere Performanzsteigerung? Gehen Sie davon aus, dass das Amdahlsche Gesetz anwendbar ist.

Hannes has written a program and, unfortunately, it is not fast enough. He knows that his program currently contains a sequential part of 40%. He has two options:

1. *Buy a new computer A with 4 processor cores and cut the running time of the sequential part by half.*
2. *Buy a new computer B with 6 Processors. He knows that B generally has a better performance than A because of its higher frequency. That shortens the execution time by another 10%.*

Provide some advise to Hannes: which of the two options provides the better performance improvement? Assume that Amdahl's law is applicable

Speedup of A ($\lambda = 0.2$, $p = 4$) according to Amdahl

$$S_A \leq \frac{1}{0.2 + 0.8/4} = \frac{1}{0.4} = 2.5$$

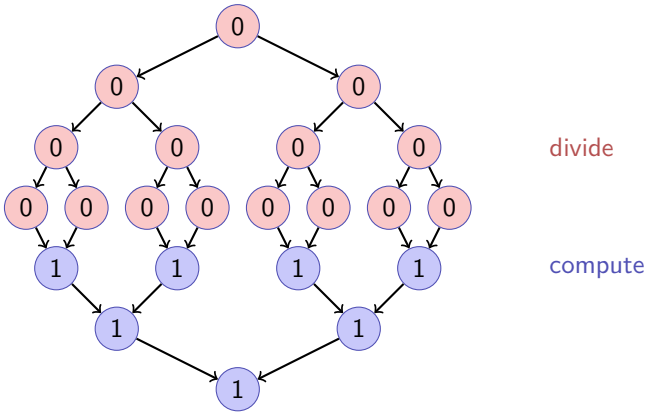
Speedup of B ($\lambda = 0.4$, $p = 6$) according to Amdahl

$$S_B \leq \frac{1}{0.4 + 0.6/6} = \frac{1}{0.5} = 2$$

Overal speedup for B : $1/0.9 \cdot S_B = 2.22 < 2.5 = S_A$
A is the better option.

/4P (b) Niklas hat ein klassisches Divide-&-Conquer Verfahren zum Finden des Minimums eines Arrays (mit Länge $n = 2^k$, $k \in \mathbb{N}$) implementiert: das Array wird sukzessive in zwei Teile zerteilt bis nur ein Element übrig bleibt. Danach werden die Resultate der zerteilten Teile verglichen und weitergereicht. Das Verfahren kann mit dem folgenden Task-Graph beschrieben werden. Wir nehmen an, dass jeder compute-Task Laufzeitkosten 1 hat und wir ignorieren die Kosten der divide-Tasks.

Niklas has implemented a classical divide&conquer procedure to find the minimum of an array (with length $n = 2^k$, $k \in \mathbb{N}$): the array is divided into two parts until only one element is left. The the results of the divided parts are compared and combined. The procedure can be described with the following task graph. We assume a compute task provides running time costs of 1 and we ignore all costs of divide tasks.



Die sequentielle Laufzeit dieses Verfahrens ist $\sum_{i=1}^{k-1} 2^i = 2^k - 1 \approx n$.
Welche Geschwindigkeitssteigerung S_p durch Parallelisierung ist unter Einsatz eines gierigen Schedulers in Abhängigkeit der Problemgrösse $n = 2^k$ und Anzahl Prozessoren p idealisiert zu erwarten? Was ist der beste zu erwartende Speedup S_∞ mit unendlich vielen Prozessoren?

The sequential running time of this procedure is $\sum_{i=1}^{k-1} 2^i = 2^k - 1 \approx n$.
What parallel speedup S_p, as a function of problem size $n = 2^k$ and number processors p can we ideally expect when we employ a greedy scheduler? What is the best possible speedup S_∞ for infinitely many processors?

Running time with p processors $T_p \leq T_1/p + T_\infty$.
$T_1 = 2^k - 1 \approx n$, $T_\infty = k = \log_2 n$
Speedup :

$$S_p \geq \frac{T_1}{T_p} = \frac{n}{n/p + \log_2 n} = \frac{1}{1/p + \log_2 n/n}$$

$$S_\infty = \frac{n}{\log_2 n}$$

Wir verwenden im folgenden Teil der Aufgabe C++ Code und nehmen an, dass Sie die jeweilige Syntax und Semantik verstehen. Es geht um das theoretische Verständnis der nebenläufigen Vorgänge. Treffen Sie insbesondere keine aussergewöhnliche Annahmen über die verwendete Architektur, das Speichermodell oder das Verhalten des Compilers.

For the following part of the task we use C++ code and assume that you understand the syntax and semantics. But this task is more about the theoretical understanding of what can happen in a concurrent setup. Do particularly not make make any exceptional assumptions about the architecture used, the memory model or the behavior of the compiler.

Es werden Threads ausgeführt. Wir gehen jeweils davon aus, dass die Funktion `print` jeweils einen Buchstaben ausgibt. Geben Sie für die Programme jeweils alle möglichen Ausgaben 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.

Threads are executed. We assume for each part that the function `print` outputs a character. For each of the programs, provide all possible outputs, separated by semicolons if necessary. 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(){
    std::thread t(A);
    t.join();
    print('C');
    print('D');
    A();
}

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

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

```
ABCDAB
```

Das Programm terminiert/*the program terminates*

√ immer/*always* ◯ nie/*never* ◯ manchmal/*sometimes*

/2P (d)

```
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);
    A();
    t.join();
    print('D');
}

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

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

```
CABABD
```

Das Programm terminiert/*the program terminates*

√ immer/*always* ◯ nie/*never* ◯ manchmal/*sometimes*

(e) /2P

```
using guard = std::unique_lock<std::mutex>;
std::mutex m; // global variable useable by all threads
std::condition_variable c;
int choose;

void A(){
    guard lock(m);
    print('A');
    c.wait(lock,[&]{return choose == 1;});
    choose = 2;
    c.notify_one();
    print('B');
}

void B(){
    guard lock(m);
    print('C');
    c.wait(lock,[&]{return choose == 2;});
    choose = 1;
    c.notify_one();
    print('D');
}

int main(){
    choose = 0;
    std::thread t1(A);
    std::thread t2(B);

    t1.join();
    t2.join();
    std::cout << "done" << std::endl;
}
```

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

```
AC CA
```

Das Programm terminiert/*the program terminates*

◯ immer/*always* √ nie/*never* ◯ manchmal/*sometimes*

/18P

Aufgabe 7: Programmieraufgabe: Teilchensimulation (18P)

Diese Aufgabe zum Thema Parallele Programmierung soll am Computer gelöst werden. Sie können sich die Zeit frei einteilen. Wir **empfehlen** aber, dass Sie **nicht mehr als 45 Minuten** für diese Aufgabe aufwenden.

Lösen Sie diese Aufgabe in der Online-Umgebung (Code Expert via Moodle).

This following part on parallel programming needs to be solved at the computer. You are free in how you divide the time. However, we **recommend to spend not more than 45 minutes** *on that problem.*

Solve this task in the online environment (Code Expert via Moodle).

Moodle-Passwort wird zu Beginnn der Prüfung bekannt gegeben / *Moodle password will be announced at the beginning of the exam*