Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prüfung **(Lösung)**

Datenstrukturen und Algorithmen (D-MATH RW)

Felix Friedrich, Departement Informatik

ETH Zürich, 26.1.2018.

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 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 spoken languages). 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 false 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:	27	15	18	15	15	16	14	120
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 (27P)

1) In dieser Aufgabe sollen nur Ergebnisse angegeben werden. Sie können sie direkt bei den Einzelteilen notieren.

   In this task only results have to be provided. You can note them down in the parts.

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.

/1P (a) Gegeben sind Daten der Länge n. Wie viele Vergleiche sind minimal nötig, um das Maximum des Datensatzes zu finden?

Given data with length n. How many comparisons are minimally required to determine the maximum of the data set?

Minimale Anzahl Vergleiche/*minimal number of comparisons*: $n - 1 \in \Theta(n)$

/3P (b) Nachfolgend sehen Sie drei Folgen von Momentaufnahmen (Schritten) der Algorithmen (a) Sortieren durch Einfügen, (b) Sortieren durch Auswahl und (c) Bubblesort. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

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

Auswahl

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

Bubblesort

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

Einfügen

(c) Im folgenden ist eine Hashtabelle dargestellt (nach vorgängigem Einfügen der Schlüssel 7, 18 und 29). Es wird offenes Double-Hashing verwendet. Die Hashfunktion ist $h(k) = k \bmod 11$ und für die Sondierung wird die Funktion $h'(k) = 1 + (k \bmod 9)$ verwendet. Für die Sondierung wird subtrahiert. Fügen Sie die folgenden Schlüssel in die (teilweise schon belegte) Hashtabelle in. Wie viele Kollisionen treten dabei auf?

In the following a hash table is displayed (after keys 7, 18 and 29 had been inserted previously). We use open double-hashing. The hash function is $h(k) = h \bmod 11$ and for probing the function $h'(k) = 1 + (k \bmod 9)$ is used. Probing works using subtraction. Insert the following keys into the (partially occupied) hash table. In doing so, how many collisions take place?

/4P

Einzufügende Schlüssel/*Keys to be inserted*: 6, 17, 19

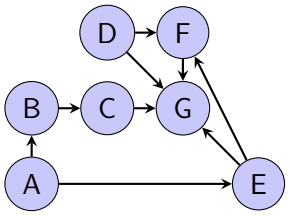
		19		29		18	7	**17**		**6**
0	1	2	3	4	5	6	7	8	9	10

Anzahl Kollisionen/*number of collisions*: ◯ 2 ◯ 3 ◯ 4 √ 5 ◯ 6

(d) Der folgende Graph wird mit Tiefensuche traversiert. Die Suche startet beim Knoten A. Geben Sie eine Reihenfolge an, in der die Knoten erreicht werden können.

The following graph is traversed using depth first search. The search starts at node A. Provide an order in which the nodes can be visited.

/2P



A,B,C,G,E,F oder A,E,F,G,B,C oder A,E,G,F,B,C

(e) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Korrekte Antworten ergeben 1 Punkt. Inkorrekte oder fehlende Antworten ergeben keinen Punkt.

Mark in the following if the statements are true or false. Correct answers provide 1 points. Incorrect or missing answers provide no point.

/4P

Jeder vergleichsbasierte Sortieralgorithmus benötigt im schlechtesten Fall $O(n \log n)$ Vergleiche.

Any comparison based sorting algorithm requires $O(n \log n)$ key comparisons in the worst case.

◯ wahr/*true* √ falsch/*false*

Werden in einen binären Suchbaum ausschliesslich Schlüssel eingefügt und keine entfernt, so entspricht die Preorder-Reihenfolge genau der Einfügereihenfolge.

If in a binary search tree keys are only inserted but none is removed, then the pre-order matches the insertion order exactly.

◯ wahr/*true* √ falsch/*false*

Der einfache Algorithmus von Dijkstra kann auch auf Graphen mit negativen Gewichten angewendet werden – unter der Voraussetzung, dass der Graph azyklisch ist.

The simple algorithm of Dijkstra can be applied to graphs with negative weights – provided that the graph is acyclic.

◯ wahr/*true* √ falsch/*false*

Jeder kreisfreie Graph kann toplogisch sortiert werden.

Any cycle-free graph can be sorted topologically.

√ wahr/*true* ◯ falsch/*false*

/2P (f) Die folgende Tabelle enthält eine Union-Find Datenstruktur zum Mengensystem $\{\{1,2,3,9\},\{4,6,7\},\{5,8\},\{10\}\}$.
Ergänzen Sie die Tabelle, so dass sie die Union-Find Datenstruktur nach der Operation Union(Find(3),Find(4)) enthält.

The following table contains a Union-Find data structure to the set of sets $\{\{1,2,3,9\},\{4,6,7\},\{5,8\},\{10\}\}$. Complement the table such that it contains the Union-Find data structure after the operation Union(Find(3),Find(4)).

[index]	1	2	3	4	5	6	7	8	9	10
initial [parent]	1	1	1	6	5	6	6	5	3	10
nach/*after* Union(Find(3),Find(4))	1|6	1	1	6	5	1|6	6	5	3	10

/3P (g) Die folgende Tabelle enthält einen Min-Heap mit 9 Einträgen (als Array gespeichert), welcher die geraden Zahlen von 2 bis 18 enthält. Geben Sie an, wie die der Min-Heap aussieht, wenn man mit dem Algorithmus der Vorlesung die Zahl 0 hinzugefügt hat.

The following table contains a Min-Heap with 9 entries (represented as array). The Min-Heap contains the even numbers from 2 to 18. Specify the min-heap after the number 0 has been inserted according to the algorithm from the lectures.

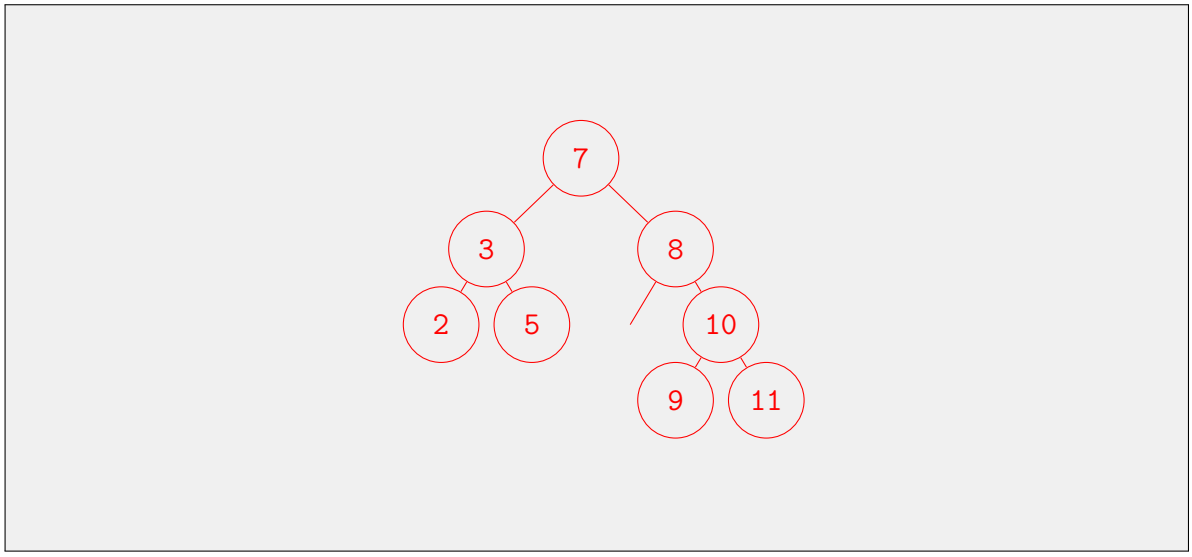
[index]	1	2	3	4	5	6	7	8	9	10
initial	2	4	6	8	10	12	14	16	18	–
nach/*after* Insert(0)	0	2	6	8	4	12	14	16	18	10

(h) Zeichnen Sie einen binären Suchbaum, dessen Post-Order Traversierung die folgende Folge ergibt:

Draw a binary search tree where a post-order traversing would provide the following sequence:

/4P

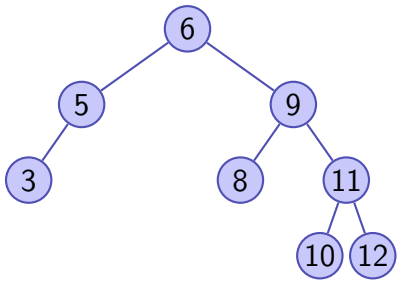
2, 5, 3, 9, 11, 10, 8, 7



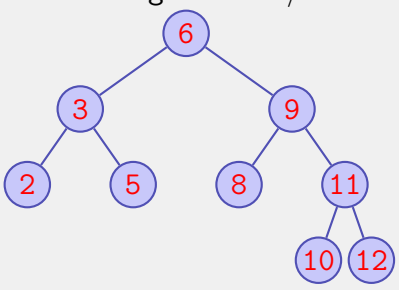
(i) Fügen Sie in untenstehendem AVL-Baum zuerst den Schlüssel 2 ein und löschen Sie danach im entstandenen AVL-Baum den Schlüssel 6.

First insert into the AVL-Tree below the key 2. After insertion, delete the key 6 from the created AVL-Baum.

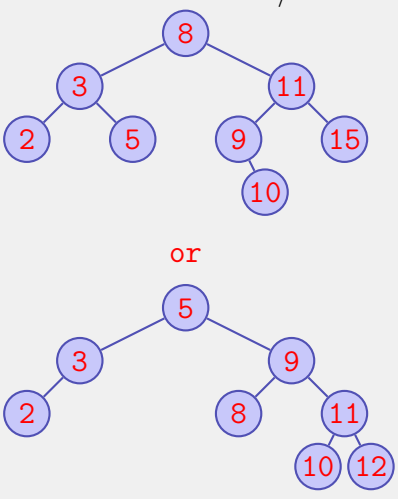
/4P



Nach Einfügen von 2 / *After insertion of 2*



Nach Löschen von 6 / *After deletion of 6*



Aufgabe 2: Asymptotik (15P)

/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 richtigen Reihenfolge, da $n^3 \in \mathcal{O}(n^5)$ und $n^5 \in \mathcal{O}(n^7)$.

Provide an order for the functions below 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 correct order because $n^3 \in \mathcal{O}(n^5)$ and $n^5 \in \mathcal{O}(n^7)$.

$$n^2 + 2n + 1, \quad n \sum_{i=0}^{n} i, \quad n \log n^n, \quad \sqrt{n} \log n, \quad n \log n^2, \quad n!$$

$\sqrt{n} \log n, \quad n \log n^2, \quad n^2 + 2n + 1, \quad n \log n^n, \quad n \sum_{i=0}^{n} i, \quad n!$

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

$$T(n) = \begin{cases} 1 + \sum_{i=0}^{n-1} T(i), & n > 0 \\ 1 & n = 0 \end{cases}$$

Geben Sie eine geschlossene (nicht rekursive), einfache Formel für $T(n)$ an und beweisen Sie diese mittels vollständiger Induktion. *Provide a closed (non-recursive), simple formula for $T(n)$ and prove it using mathematical induction.*

The following equation holds for $n > 0$ (yes, also for $n = 1$)

$$\begin{aligned} T(n) &= 1 + \sum_{i=0}^{n-1} T(i) = 1 + T(n-1) + (1 + \sum_{i=0}^{n-2} T(i)) - 1 \\ &= 2 \cdot T(n-1) \\ &= 2^2 \cdot T(n-2) = \ldots = 2^n \cdot T(0) = 2^n \end{aligned}$$

Induktion:

1. Hypothese $T(n) = 2^n$.

2. Anfang $T(0) = 1$.

3. Schritt $(n-1 \to n)$: $T(n) = 1 + \sum_{i=0}^{n-1} 2^i = 1 + 2^n - 1 = 2^n$

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){
    for (int i = 0; i<n ; ++i ){
        for (int j=0; j<i; ++j){
            f()
        }
    }
}
```

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

$\Theta(n^2)$

/2P (d)

```
void g(int n){
    if (n > 1){
        g(n/2);
    }
    f();
}
```

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

$\Theta(\log(n))$

/3P (e)

```
void g(int n){
    for (int i = 0; i<n ; ++i ){
        g(i)
    }
    f();
}
```

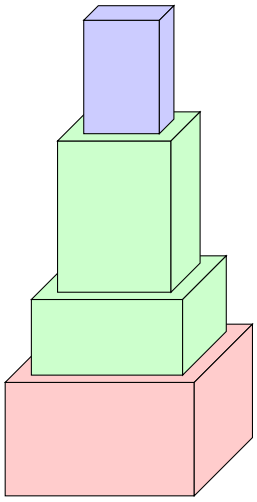
Anzahl Aufrufe von f / *Number of calls of f*

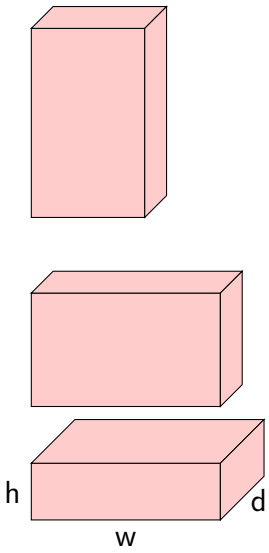
$\Theta(2^n)$

Aufgabe 3: Dynamic Programming (18P)

In dieser Aufgabe geht es darum, Boxen zu stapeln. Man geht normalerweise davon aus, dass es genügend Boxen jeder Sorte gibt, so dass jede Box in jeder Orientierung verfügbar ist (Abbildung rechts). Um die Aufgabe nicht unnötig schwierig zu machen, abstrahieren wir jedoch von dieser Tatsache und betrachten im folgenden eine Box immer mit Orientierung: sie darf also nicht gedreht werden.

Objective of this task is to stack boxes. It is usually assumed that there are enough boxes of a kind such that each box is available in all orientations (right hand side of the figure below). However, to not make the task unnecessary difficult, we abstract away this fact and consider each box with orientation: it must not be rotated.





Gegeben: Menge B von n Boxen $[w_i \times d_i \times h_i]$, $1 \leq i \leq n$, jeweils mit Breite $w_i > 0$, Tiefe $d_i > 0$, $w_i \geq d_i$ und Höhe $h_i > 0$.
Aufgabe: Konstruiere einen Stapel Boxen mit maximaler Höhe. Dabei darf Box i nur auf Box j gestapelt werden, wenn $w_i < w_j$ und $d_i < d_j$ gilt (siehe Abbildung links).

Given: set B with n boxes $[w_i \times d_i \times h_i]$, $1 \leq i \leq n$, each with width $w_i > 0$, depth $d_i > 0$, $w_i \geq d_i$ und height $h_i > 0$.
Task: construct a stack of boxes with maximal height. Here the box i may only be stacked on box j if $w_i < w_j$ and $d_i < d_j$ (cf. left figure).

Beispielszenario mit sechs orientierten Boxen:

Example scenario with sixes oriented boxes:

Box	1	2	3	4	5	6
$[w \times d \times h]$	$[1 \times 2 \times 3]$	$[1 \times 3 \times 2]$	$[2 \times 3 \times 1]$	$[3 \times 4 \times 5]$	$[3 \times 5 \times 4]$	$[4 \times 5 \times 3]$

Geben Sie einen Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die Höhe des höchstmöglichen Stapels berechnet.
(Der triviale Algorithmus, der alle Lösungen aufzählt, ergibt keine Punkte.)

Provide a dynamic programming algorithm for computing the minimum cost (as described above). Address the following aspects in your solution.
(The trivial algorithm that simply enumerates all possible solutions does not give any points.)

/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

Alternative: array of length n, topologically sorted by the relation 'can be stacked on top of'

Bedeutung Eintrag / *entry meaning*:

Entry at row i and column j: height of highest possible stack formed from maximally i boxes and basement box j.

Alternative: height of highest possible stack formed with basement box 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?

First row: $t_{1,c} = h_c$ for each $1 \leq c \leq n$.

Other rows: $t_{r,c} = h_c + \max\{t_{r-1,j} : 1 \leq j \leq n, w_j < w_c, d_j < d_c\}$. Here we define $\max\{\} = 0$. As a technical trick / sentinel we may alternatively add an artificial column with box $[0 \times 0 \times 0]$.

alternatively: provide a topological sorting of the boxes (half-order), for example order them ascending left to right in the array. Compute maximal height from right to left.

(III)

In welcher Reihenfolge können die Einträge berechnet werden?

In which order can the entries be computed?

row-wise increasing with any column-order.

alternative: Compute maximal height from right to left.

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

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

Maximum in row n.

alternative: maximum in first row

(b) Zusätzlich zur maximalen Höhe wollen Sie auch angeben, welche Boxen dafür gestapelt werden. Beschreiben Sie detailiert, wie Sie das bewerkstelligen.

In addition to the maximum height you want to provide the boxes used. Describe in detail how this is done.

/4P

We can either keep a second $1 \times n$ table with the optimal predecessor and update that during computation of the maximal height. Or we walk back the table using the max-identity above.
alternative: store the topological order of the boxes in a graph and memorize best way. Or walk back the array using the topological sort and heights.

(c) Geben Sie nun eine höchsten Stapel für obiges Beispielszenario an. Wie hoch wird der Turm? Geben Sie die Tabelle an.

Now provide the highest possible stack of boxes for the example scenario above. What is the maximum height? Provide the table.

/3P

$[w \times d]$	$[1 \times 2]$	$[1 \times 3]$	$[2 \times 3]$	$[3 \times 4]$	$[3 \times 5]$	$[4 \times 5]$	
1	$\underline{3}$	2	1	5	4	3	
2	3	2	$\underline{4}$	8	8	8	Turmhöhe 12.
3	3	2	4	$\underline{9}$	8	11	
4	3	2	4	9	8	$\underline{12}$	

(d) Geben Sie die Laufzeiten für die Berechnung der maximalen Höhe und des Stapels in Θ-Notation möglichst knapp mit Begründung an.

Provide the run-times for the computation of the maximal heightin Θ notation as tight as possible with short explanation.

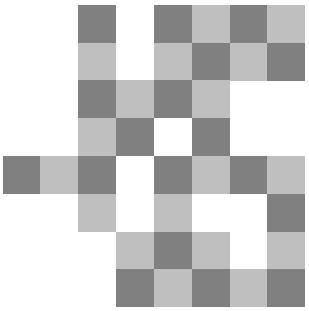
/3P

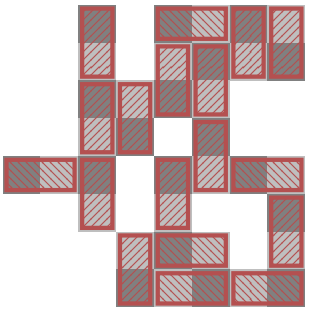
Bestimmung der Tabelle: $\Theta(n^3)$, für jeden Eintrag müssen alle Einträge der vorherigen Zeile durchlaufen werden, also $\Theta(n^2)$ pro Zeile.
Berechnung der optimalen Lösung durch Rückverfolgung im schlechtesten Fall $\Theta(n^2)$, da von jedem k das nächste k gesucht werden muss.
Alternativ $\Theta(n)$, wenn eine zweite Tabelle geführt wird.
Alterantivlösung: Topologisches Sortieren in $\Theta(n^2)$. Durchlaufen von rechts nach links in $\Theta(n)$, insgesamt $\Theta(n^2)$. Rückverfolgung auch $\Theta(n^2)$ oder mit memorisieren: $\Theta(n)$

Aufgabe 4: Flussprobleme (15P)

Gegeben sei ein $n \times n$ Schachbrett, von dem einige Felder entfernt wurden. Die verbliebenen Felder sollen mit Dominosteinen belegt werden. Ein Dominostein kann ein Paar von vertikal order horizontal benachbartes Feldern belegen. Beschreiben Sie einen möglichst effizienten Algorithmus, mit dem Sie feststellen können, ob das verbleidende Brett vollständig überdeckt werden kann. Dabei muss jedes nicht entfernte Feld vom Schachbrett von genau einem Dominostein bedeckt sein und jeder Dominostein muss genau zwei nicht entfernte Felder bedecken.

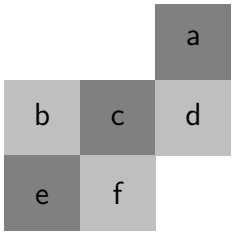
Let an $n \times n$ chessboard be given. Some of the squares have been removed. The remaining squares shall be covered with dominoes. A domino can cover either a vertical or horizontal pair of neighboring squares. Describe an efficient algorithm that you can use in order to find out if the remaining board can be completely covered. Each remaining square must be covered by one and only one domino and each domino must cover two remaining squares.



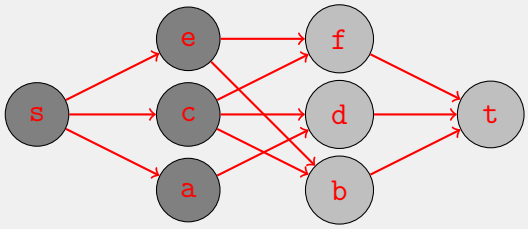


/10P (a) Modellieren Sie obiges Problem als Flussproblem. Geben Sie einen möglichst effizienten Algorithmus an, mit dem Sie bestimmen können, ob die Felder mit Dominosteinen überdeckt werden können. Illustrieren Sie Ihren Algorithmus an folgendem Beispiel (Graph angeben).
Tipp: Sie können die Schwarz-Weiss Färbung des Schachbretts ausnutzen, um damit Zyklen von Dominosteinen (und somit Mehrfachbelegung von Feldern) zu vermeiden.

Model the problem described above as a flow problem. Specify an efficient algorithm that can be used in order to determine if the squares can be covered with dominoes. Illustrate your algorithm using the following example (provide the graph).
Hint: you can make use of the black-white coloring of the chessboard in order to avoid cycles of dominoes (and thus having more than one domino on a square).



Bipartite matching problem: connect black fields to neighbouring white fields with the condition that each field is used only once.
Directed graph $G = (V, E, c)$ with V: set of fields plus artificial source s and sink t. Edges from black sites to neighbouring white sites, edges from source s to all black sites and from all white sites to the sink t. $c_{ij} = 1$ for all sites.
Use the Ford-Fulkerson Algorithmus to determine the maximum flow from s to t. If the maximal flow equals number of black (and white) sites, then the chessboard can be covered with dominos as required.
Example with field above:



(b) Geben Sie die Laufzeit Ihres Algorithmus (im schlechtesten Fall) in Abhängigkeit von der Seitenlänge n des Brettes an. Gehen Sie davon aus, dass vom ursprünglichen Schachbrett nicht mehr als die Hälfte Felder gelöscht wurden.

Provide the running time of your algorithm (in the worst case) as a function of the side-length n of the board. You can assume that from the chessboard not more than half of the squares have been removed.

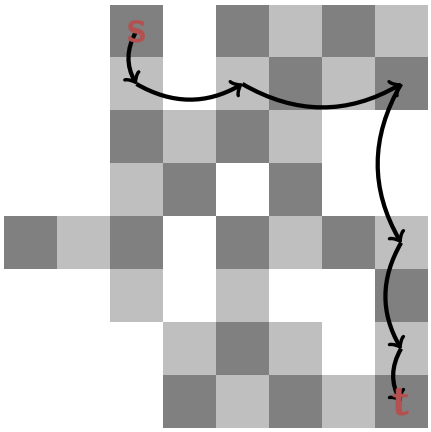
/5P

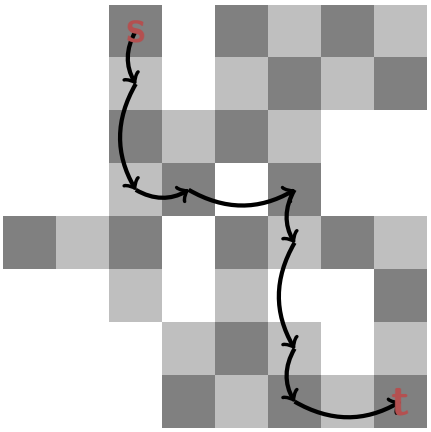
Number edges is $\Theta(n^2)$ because there is $\Theta(n^2)$ vertices (cites) and each cite has maximally 4 neighbours. Therefore the algorithm terminates after $O(|f_{max}| \cdot |E|) = O(|V| \cdot |E|) = O(n^4)$, steps.

Aufgabe 5: Kürzeste Wege (15P)

Gegeben sei ein $n \times n$ Schachbrett, von dem einige Felder entfernt wurden. Eine Figur darf auf dem Schachbrett entweder horizontal oder vertikal in beliebiger Weite (beschränkt durch die Seitenlänge n des Schachbretts) springen. Es gibt jedoch folgende Einschränkungen: die Figur darf bei jedem Sprung maximal eine Einheit weiter oder näher springen als beim vorigen Sprung. Die Richtung spielt für diese Einschränkung keine Rolle. Beim ersten Sprung darf sie genau eine Einheit weit springen. Die Figur muss immer auf einem nicht entfernten Feld landen.
Beschreiben Sie einen effizienten Algorithmus mit dem Sie einen Weg mit minimalen Anzahl Sprüngen von einem Startfeld s zu einem Zielfeld t finden können.
In der linken Abbildung ist eine erlaubte Sprungfolge von s nach t mit 6 Sprüngen zu sehen, in der rechten Abbildung benötigt die Figur 8 erlaubte Sprünge.

Given an $n \times n$ chessboard where some of the squares have been removed. A piece can move on the chessboard: it can either jump horizontally or vertically arbitrarily far (obviously limited by the side lenght n of the chessboard) with the following restrictions: the piece can jump maximally one unit further or closer than in the previous jump. The direction does not play any role for this limitation. The first jump has range one. The piece must land on a non-removed field each time.
Describe an efficient algorithm that you can use in order to find a way with minimal number of jumps from a start square s to some target square t.
In the left figure a permitted sequence of jumps from s to t with 6 jumps is displayed. The right figure contains a permitted sequence of eight jumps from s to t.





/5P (a) Modellieren Sie den Zustandsraum als gerichteten Graphen. Beschreiben Sie möglichst genau (aber informell) welche Knoten von einem gegebenen Knoten erreichbar sind. Tipp: ein Feld des Schachbrettes kann in der Knotenmenge des Graphen durchaus mehrfach vorkommen.

Model the state space as a directed graph. Describe as precisely as possible (but informally), which nodes can be reached from a given node. Hint: a square of the chessboard can be present multiple times in the set of nodes of the graph.

The nodes consist of a replication of the ckecker board squares where each node is additionally equipped with a current speed value. Edges are between nodes in the same row or column, with maximally speed distance and speed value of speed-1 to speed+1.

(b) Entwerfen oder nennen Sie einen möglichst effizienten Algorithmus für das oben beschriebene Problem. Beschreiben Sie den Algorithmus in kurzen Worten. Der Algorithmus soll die minimale Anzahl Sprünge ausgeben oder eine Meldung, wenn kein Weg zum Ziel existiert.

Design or name an efficient algorithm for the problem described above. Describe the algorithm shortly. The algorithms shall output the minimal number of jumps or a message if there is no path from s to the target t.

/5P

This is a classical breadth first search problem: keep a fifo-list of nodes that are currently being visited, start with $(s, 1)$. Mark each visited field as visited such that it is not visited twice. In order to keep track of the number of jumps either the fields can be marked with minimal number of jumps or the fifo-list needs to be swapped for each new wavefront of the algorithm.

(c) Geben Sie die Laufzeit des Algorithmus in Abhänigkeit von n an.

Provide the running time of the algorithm as a function of n.

/5P

Breadth first search takes $O(|V| + |E|)$. Here $|V| = O(n^3)$, n^2 for the cites and n for the jump width. Each node has maximally 12 neighbours (three in each direction), i.e. $|E| = O(|V|)$, thus the algorithm takes n^3 steps.

Aufgabe 6: Parallele Programmierung (16P)

/6P (a) Die Gesetze von Amdahl und Gustafson sind Modelle der Laufzeitverbesserung bei Parallelisierung (Amdahl geht von einem festen relativen sequentiellen Anteil aus, während Gustafson von einem festen absoluten sequentiellen Teil ausgeht). In praktischen Anwendungen stimmen beide Modelle nicht genau.
Welches der beiden Modelle sagt die Laufzeitverbesserung für das folgende Beispiel besser voraus? Wir gehen davon aus, dass im Falle von $n = 10$ der sequentielle Anteil bei 20% liegt. Begründen Sie Ihre Antwort.

The laws of Amdahl and Gustafson are models of speedup for parallelization. (Amdahl assumes a fixed relative sequential portion, Gustafson assumes a fixed absolute sequential part). In practical applications, the two models do not exactly match.
Which of the two models provides a better estimate of the speedup in the following example? We assume that for $n = 10$ the sequential part is 20% of the overal work. Justify your answer.

Anzahl Prozessoren *number processors*	Problemgrösse *problem size*	Laufzeit *running time*
$p = 1$	$n = 10$	$t = 100$ ms
$p = 8$	$n = 100$	$t = 160$ ms

Observed Speedup:

$$S = 1000/160 = 6.25$$

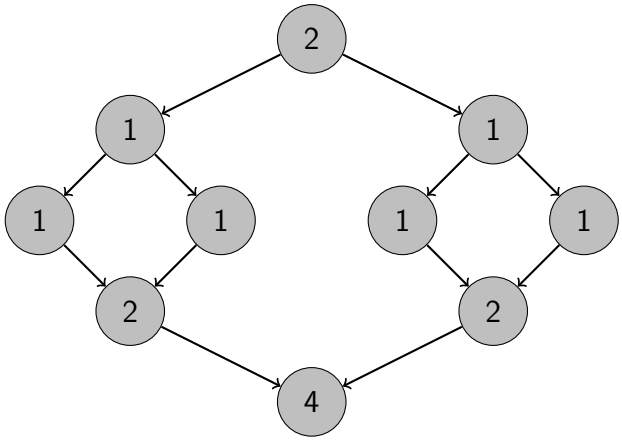
Speedup according to Amdahl or Gustafson

$$S_A = \frac{1}{\lambda + (1-\lambda)/p} = 10/3 = 3.\overline{3}$$
$$S_G = \lambda + (1-\lambda) \cdot p = 6.6$$

Gustafson is closer.

/4P (b) Geben Sie für folgenden Taskgraphen eine möglichst kleine obere Schranke an für die Ausführungszeit und eine möglichst grosse untere Schranke an für den Speedup den Sie erhalten, wenn Sie bei Einsatz eines beliebigen gierigen Schedulers beliebige viele Prozessorkerne (S_∞) oder 4 Prozessorkerne (S_4) einsetzen. Die Zahlen in den Knoten bezeichnen die jeweilige Ausführungszeit [ms].

Provide for the following task graph a smallest possible upper bound for the execution time and a largest possible lower bound for the speedup that you observe when an arbitrary greedy scheduler is employed and you have arbitrarily many processors (S_∞) or 4 processors (S_4) available. The number in the nodes provide execution time [ms].



$T_\infty \leq$ 10 ms

$T_4 \leq$ 16/4+10 = 14 ms [1]

$S_\infty \geq$ 16/10 = 1.6

$S_4 \geq$ 16/14= 1.14

(c) Erklären Sie kurz / in Stichworten einen Algorithmus zur Erkennung von Deadlocks. Gehen Sie kurz auf den Algorithmus auf dem Graphen ein. Technische Implementationsdetails sind nicht gefragt.

Explain shortly / using keywords an algorithms to detect deadlocks. Shortly explain the algorithm on the graph. Technical implementation details are not required.

/4P

Detect deadlocks as cycles in the dependency graph. Cycles in a graph can be detected with a depth first search: check and set a visited flag on each node.

(d) Erklären Sie kurz / in Stichworten eine Strategie zur Vermeidung von Deadlocks.

Explain shortly / using keywords a strategy to avoid deadlocks.

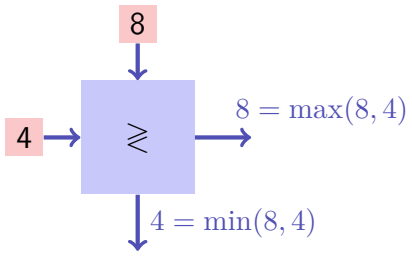
/2P

Deadlocks are avoided by avoiding the dependency cycles, e.g. by ordering the resources.

[1] the shape and node values of the special graph above also allows for $T_4 \leq T_\infty = 10ms$, which also would have resulted in full points. The task graph was chosen unfortnuate. In a better version, the 1 and 2 node on the left hand side of the graph should have been exchanged.

/14P

Aufgabe 7: Parallele Programmierung (14P)



Ein Sortierknoten (Comparator) ist im folgenden ein Objekt mit zwei Eingabeports und zwei Ausgabeports. Wir nehmen an, dass mit jedem Eingabeport ein Thread verbunden ist, welcher dort Daten mit `put` eingibt. Ebenso nehmen wir an, dass mit jedem Ausgabeport ein Thread verbunden ist, welcher dort Daten mit `get` ausliest. Es gilt

- Threads, die die Resultate am Ausgabeport mit `get` lesen, können nur fortfahren, wenn an beiden Eingabeports ein Wert anliegt.
- Threads, welche am Eingabeport Werte mit `put` anlegen wollen, können nur fortfahren, wenn der vorige Wert nicht mehr gebraucht wird, also an beiden Ausgabeports die Werte (mit `get`) abgeholt wurden.
- Am Ausgabeport 0 wird das Minimum bereitsgestellt, am Ausgabeport 1 das Maximum.

Vervollständigen Sie nachfolgenden Code so, dass obige Semantik implementiert wird.

In the following, a comparator is an object with two input ports and two output ports. We assume that at each input port there is a thread that provides data using function `put`. Furthermore, we assume that at each output port there is a thread that reads data using function `get`. It holds that

- *threads that read results at the output port using `get` can only continue when on each input port there is a value available.*
- *threads that provide values at the input ports using `put` can only continue executing when the previous value is not used any more, i.e. when the values have been used from both output ports (via `get`).*
- *At output port 0 the minimum is made available, on output port 1 the maximum.*

Complement the following code such that the semantics from above are implemented.

```
// includes and shortcuts on this page in order to save space

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

```
// program continues on the right hand side
```

```
class Comparator{
  private:
    mutex mtx;
    condition_variable cv;
    int values[2]; // recently entered values at input ports 0 and 1
    // available[x][y]: value at input port x available for output port y
    bool available[2][2]={{false,false},{false,false}};
  public:
    // pre: port = 0 or 1
    // post: as soon as the port is free, set its value
    void put(int port, int value){
        unique_lock lock(mtx);
        cv.wait(lock, [&]
        {return !available[port][0] && !available[port][1];});

        2p for lock,3p for correct condition

        values[port] = value;
         // value on this input port is now available for both output ports
        available[port][0] = available[port][1] = true;
        cv.notify_all();

        2p for notify all
    }
    // pre: port = 0 or 1
    // post: as soon as both inputs are available, return minimum or maximum
    int get(int port){
        unique_lock lock(mtx);
        cv.wait(lock, [&]
        {return available[0][port] && available[1][port];});

        2p for lock,3p for correct condition

        // input ports not required for output at this port any more
        available[0][port] = available[1][port] = false;
        cv.notify_all();

        2p for announcement with notify all
        if (port == 0){
            return std::min(values[0],values[1]);
        } else {
            return std::max(values[0],values[1]);
        }
    }
```

```
};
```