Instruktionen zur Prüfung Algorithmen & Datenstrukturen

Ablauf

- **Dauer und Gewichtung:** Die Dauer der Prüfung beträgt 4 Stunden. In dieser Zeit müssen die Theorie- und die Programmieraufgaben gelöst werden. Beide Teile sind auf je 2 Stunden Dauer ausgelegt und zählen gleich viel. Trotzdem können Sie individuell mehr oder weniger Zeit pro Teil verwenden. Wir empfehlen Ihnen dringend, mit dem Programmierteil zu beginnen, und nach spätestens 2 Stunden mit dem Theorieteil fortzufahren.
- **Frühe Abgabe:** Aufgrund der Prüfungsstruktur ist eine frühzeitige Abgabe nicht möglich.

Prüfungsbedingungen

- **Störungen:** Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- **Disziplinarordnung:** Betrugsversuche unterstehen den Strafnormen der Disziplinarordnung der ETH und können zur Exmatrikulation führen.
- **Fragen:** Während der Prüfung werden inhaltliche Fragen ausschliesslich über den Judge beantwortet (siehe Punkt 4 der technischen Anleitung). Das gilt auch für die Theorieaufgaben.

Logistik

- **Erlaubte Hilfsmittel:** Ausser Stiften, Wörterbüchern und einer einfachen Uhr sind keine Hilfsmittel erlaubt (keine kommunikationsfähigen, programmierbaren und/oder speicherfähigen Geräte). Vor Beginn der Prüfung sind alle nicht erlaubten Gegenstände wegzuräumen. Jegliche elektronischen Geräte, insbesondere Natels, Smartwatches und Passwort-Sticks sind auszuschalten, dürfen auf keinen Fall benutzt werden und müssen ins Gepäck.
- **Gepäck und Jacken** müssen am Rand der Räume gelagert werden und dürfen nicht mit an den Arbeitsplatz genommen werden. Wir können Ihr Gepäck nicht beaufsichtigen, Sie deponieren es also auf eigenes Risiko, und wir empfehlen Ihnen deshalb keine Wertsachen zur Prüfung mitzubringen. Beschriften Sie Ihre Gegenstände, um Verwechslungen vorzubeugen.
- **Legi-Kontrolle:** Legen Sie Ihre Legi gut sichtbar vor sich auf den Tisch. Bei der Kontrolle nach Prüfungsbeginn werden wir Sie auch bitten, die Submissions-Webseite auf dem Judge zu öffnen. Führen Sie daher gleich nach Beginn die Schritte der technischen Anleitung durch.
- **Essen und Getränke** am Arbeitsplatz sind in vernünftiger Ausprägung erlaubt. Dabei dürfen aber andere Prüflinge nicht durch Gerüche oder Geräusche gestört werden. Erlaubt sind nur trockene (keine flüssigen oder fettigen) Speisen, und Getränke müssen in wiederverschliessbaren Verpackungen aufbewahrt und verschlossen werden, wenn nicht getrunken wird. Bitte sehr vorsichtig sein und hinterher aufräumen.
- **Schallschutz:** Nur Ohropax (oder ein vergleichbares Konkurrenzprodukt) ist erlaubt, kein Kapselgehörschutz, keine Kopfhörer.
- **Toilette:** Wenn Sie während der Prüfung die Toilette benutzen möchten, melden Sie sich per Handzeichen. Eine Aufsichtsperson begleitet Sie. Im Schleusenraum ist es **nicht** möglich, das WC zu benutzen. Es empfiehlt sich, als Prüfling des zweiten Durchgangs das WC vor dem Einlass in den Schleusenraum zu benutzen.

Programmierteil (Computerprüfung)

- **nethz Passwort:** Zur Anmeldung am Judge benötigen Sie Ihren nethz-Account. Stellen Sie sicher, dass Sie sich an Ihre Logindaten erinnern können.
- **Beste Abgabe zählt:** Genau wie bei den Programmieraufgaben im Semester können Sie die Programmieraufgaben so oft einschicken, wie Sie wollen. Pro Aufgabe zählt die beste Einsendung. Es zählt nur, was auf dem Judge eingereicht wurde.
- **Nicht ausschalten!** Der Computer darf auf keinen Fall ausgeschaltet werden, auch nicht gegen Ende der Prüfung. Es droht der Verlust aller Daten!
- **Bei Problemen:** Falls ein Systemfehler auftritt, melden Sie sich bitte sofort durch Handzeichen und klicken Sie die entsprechende Meldung nicht weg.
- **Java-Bibliotheken:** Sie dürfen nur die Objekte, Funktionen etc. von `java.lang.*` benutzen, und keine anderen Pakete. Das Paket `java.lang` enthält alle Objekte, die standardmässig verfügbar sind, zum Beispiel Math, Integer, String, System, Double, Boolean, etc., und Sie dürfen diese frei verwenden (z.B. `Math.max()` oder `Integer.MAX_VALUE` sind erlaubt).

  Hingegen ist das Importieren oder Benutzen anderer Pakete verboten, z.B. dürfen Sie `java.util.*`, `java.io.*` etc. nicht benutzen. Beachten Sie, dass dies auch die Java Collections verbietet. Das Modul `java.util.Scanner`, welches im gegebenen Template benutzt wird, um die Eingabe zu lesen, ist die einzige erlaubte Ausnahme.

  Beachten Sie auch, dass diese Restriktion nicht beim Einsenden durch den Judge geprüft wird, aber nach der Prüfung überprüft und gegebenenfalls bestraft wird.

  Das dient nicht dazu, Ihre Aufgabe schwerer zu machen – im Gegenteil (da Sie alle verbotenen Dinge nicht kennen müssen). Es dient lediglich dazu, die Lösungen auf diejenigen Dinge zu beschränken, welche Sie im Rahmen dieser Vorlesung gesehen und verwendet haben.

Theorieteil (Papierprüfung)

- Bitte schreiben Sie Ihre Studierenden-Nummer auf **jedes** Blatt.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt oder schreiben Sie Ihre Lösung direkt auf das Aufgabenblatt (insbesondere bei Aufgabe T1).
- Eigenes Papier darf nicht verwendet werden. Wir stellen genügend Papier zur Verfügung.
- Bitte schreiben Sie **lesbar** mit **blauer oder schwarzer**, nicht-löschbarer Tinte. Wir werden insbesondere nichts bewerten, was wir nicht lesen können. Die Benutzung von Bleistiften ist nicht erlaubt.
- Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden. Formulieren Sie Ihre Lösungen nachvollziehbar.
- Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Algorithmen & Datenstrukturen" verwenden, sind Erklärungen oder Begründungen nicht notwendig. Falls Sie jedoch andere Methoden benutzen, müssen Sie diese **kurz** soweit erklären, dass Ihre Ergebnisse verständlich und nachvollziehbar sind.
- Legen Sie am Ende der Prüfung alle Blätter ausser demjenigen mit Ihrem Namenssticker in das Kuvert und verschliessen Sie es.

/ 20 P

Programmieraufgabe P1.

Passwort für Einschreibung: `blumenwiese`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

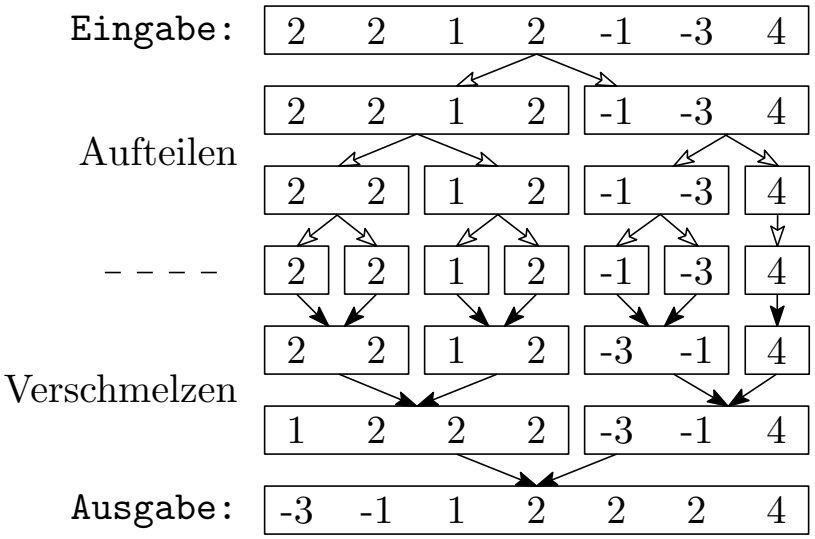
Mergesort implementieren

Ihre Aufgabe besteht in der Implementierung der Verschmelzung zweier sortierter Teilfolgen im Mergesort-Algorithmus. Ein Grossteil der Implementierung des Mergesort-Algorithmus wird bereits durch die Vorlage zur Verfügung gestellt (Einlesen der Eingabe, rekursive Aufrufe, Ausgabe). Ihre Aufgabe ist es, den Code der Funktion `mergeSort()` zu vervollständigen: Die rekursiven Aufrufe von `mergeSort()` geben zwei sortierte Arrays L und R zurück, und Ihre Aufgabe ist es, diese beiden Arrays L und R zu einem sortierten Array B zu verschmelzen (merge), das dann zurückgeliefert wird. Weitere Details finden Sie in der Vorlage.

Um die Aufgabe zu lösen, ist es ausreichend, den `TODO`-Kommentar in der Vorlage durch Ihren Java-Code zu ersetzen. Wir empfehlen dringend, den Rest der Vorlage nicht zu verändern (auch wenn es nicht verboten ist). Beachten Sie, dass die Funktion `mergeSort()` einige Informationen ausgibt, die dem Judge erlauben zu verifizieren, dass Mergesort wie beabsichtigt funktioniert, und dass diese Ausgaben exakt so verbleiben müssen.

Die zu sortierenden Zahlen sind Ganzzahlen zwischen $-1\,000\,000$ und $1\,000\,000$. Zahlen können mehrmals auftreten, und alle Zahlen müssen erhalten bleiben. Das zu sortierende Array kann bis zu 50 000 Elemente enthalten.

Beispiel Das folgende Diagramm illustriert Mergesort auf der Eingabe `2 2 1 2 -1 -3 4`.



Bewertung Sie können bis zu 100 Punkte am Judge bekommen. Ihre Implementierung der Merge-Operation sollte k Elemente stets in Zeit $\mathcal{O}(k)$ verschmelzen. Reichen Sie nur Ihr `Main.java` ein.

Instruktionen Wir stellen für diese Aufgabe eine Programmvorlage im Eclipse-Workspace zur Verfügung; die Programmvorlage implementiert bereits den Grossteil der Funktionen. Ihre Aufgabe besteht darin, den Code der Funktion `mergeSort()` zu vervollständigen.

Wie üblich enthält die Vorlage Testdaten, damit Sie lokal testen können, und enthält ein Programm `Judge.java`, das Ihr `Main.java` mit allen verfügbaren Testdaten testet – öffnen und starten sie dazu einfach `Judge.java` im Projekt. Die lokalen Testdaten unterscheiden sich von den Testdaten, welche der Online-Judge verwendet.

Die Ein- und Ausgabe werden von der Vorlage verarbeitet – Sie sollten den Rest dieses Texts nicht benötigen.

Eingabe Die erste Zeile der Eingabe enthält einzig die Anzahl der Tests.

Jeder Test besteht aus zwei Zeilen. Die erste Zeile enthält n, die Anzahl der zu sortierenden Ganzzahlen, und die zweite Zeile enthält n Ganzzahlen, die sortiert werden sollen.

Ausgabe Bei jeder Rückgabe von `mergeSort()` für mindestens 2 Elemente wird die Länge des Arrays, sowie der kleinste und der grösste Wert in einer separaten Zeile und durch Leerzeichen getrennt ausgegeben (Aufrufe von `mergeSort()` für 1 Element geben nichts aus.) Nachdem die Eingabe eines Tests sortiert wurde, wird das sortierte Array in einer einzigen Zeile durch Leerzeichen getrennt ausgegeben.

Beispiel-Eingabe (für das Beispiel oben):

```
1
7
2 2 1 2 -1 -3 4
```

Für die Beispielausgabe, betrachten Sie die Datei `testdata/example.out` *im Projekt.*

Platz für Ihre Notizen. Diese werden nicht bewertet. Nur was auf dem Judge eingereicht wird, zählt für diese Aufgabe.

/ 20 P

Programmieraufgabe P2.

Passwort für Einschreibung: `blumenwiese`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Mobilfunkmasten

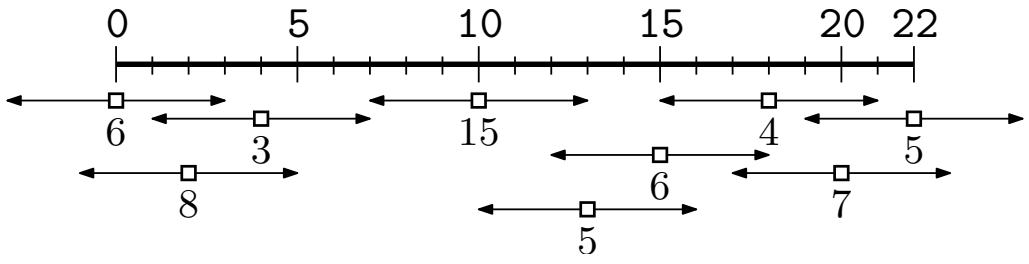
Bereits kurz nach der Fertigstellung einer neuen Autobahn beklagen sich die Fahrer/innen über die schlechte Mobilfunkabdeckung entlang der Strasse. Ihre Aufgabe ist es, neue Mobilfunkmasten zu errichten, um volle Funkabdeckung entlang der Strasse zu erreichen, und dies bei kleinstmöglichen Kosten.

Die Strasse ist gerade, L Kilometer lang, beginnt bei Kilometer 0, endet bei Kilometer L, und verfügt zu Beginn über keine Mobilfunkmasten. Jeder gebaute Mobilfunkmast deckt R Strassenkilometer in jede der beiden Richtungen ab. Eine Liste von n möglichen Mastenpositionen und der entsprechenden Kosten ist gegeben. Die i-te mögliche Mastenposition ist d_i Kilometer vom Beginn der Strasse entfernt und hat Kosten c_i für den Mastenbau, $i = 0, \ldots, n-1$. Die möglichen Mastenpositionen sind so geordnet, dass $0 \leq d_0 < d_1 < d_2 < \cdots < d_{n-1} \leq L$ gilt. Alle diese Positionen d_i sind verschieden.

Für die Eingaben gilt $1 \leq L \leq 1\,000\,000$, $1 \leq R \leq 1000$, $1 \leq n \leq 200\,000$, $1 \leq c_i \leq 1000$. Die Summe aller Kosten passt in eine Variable vom Typ `int`. Wir garantieren, dass eine Signalabdeckung der gesamten Strasse möglich ist.

Beachten Sie, dass approximative (fast optimale) Lösungen und "gierige Algorithmen" vermutlich *keine* Punkte erhalten – nur die optimale Lösung wird akzeptiert. Wir empfehlen Ihnen, ein eindimensionales dynamisches Programm zu entwerfen.

Beispiel Das folgende Bild zeigt ein Beispiel mit $L = 22$, $R = 3$ und $n = 9$, die Quadrate sind mögliche Mastenpositionen entlang der Strasse, die Zahlen darunter sind die Kosten und die Pfeile geben deren Reichweiten an. Eine optimale Lösung hat Kosten 37 (und benutzt die Masten mit Kosten $6 + 3 + 15 + 6 + 7$). Die Eingabedaten finden Sie unten.



Bewertung Sie können bis zu 100 Punkte am Judge bekommen. Für die volle Punktzahl soll Ihr Programm in Zeit $\mathcal{O}(nR)$ laufen, aber langsamere Lösungen können Teilpunkte erzielen. Reichen Sie nur Ihr `Main.java` ein.

Instruktionen Wir stellen für diese Aufgabe eine Programmvorlage im Eclipse-Workspace zur Verfügung, die Ihnen hilft die Eingabe zu lesen und die Ausgabe zu schreiben.

Wie üblich enthält die Vorlage Testdaten, damit Sie lokal testen können, und enthält ein Programm `Judge.java`, das Ihr `Main.java` mit allen verfügbaren Testdaten testet – öffnen und starten sie dazu einfach `Judge.java` im Projekt. Die lokalen Testdaten unterscheiden sich von den Testdaten, welche der Online-Judge verwendet, und sind im allgemeinen kleiner.

Die Ein- und Ausgabe werden von der Vorlage verarbeitet – Sie sollten den Rest dieses Texts nicht benötigen.

Eingabe Die erste Zeile der Eingabe enthält einzig die Anzahl der Tests.

Die erste Zeile jedes Tests enthält die Ganzzahlen L, R und n, getrennt durch Leerzeichen. Die zweite Zeile jedes Tests enthält die n möglichen Mastenpositionen als durch Leerzeichen getrennte Ganzzahlen. Die dritte Zeile jedes Tests enthält die Kosten der n Positionen als durch Leerzeichen getrennte Ganzzahlen.

Ausgabe Geben Sie für jeden Test eine Ganzzahl auf einer separaten Zeile aus: die Kosten einer optimalen Mastenplatzierung.

Beispieleingabe (für das Beispiel oben):

```
1
22 3 9
0 2 4 10 13 15 18 20 22
6 8 3 15 5 6 4 7 5
```

Beispielausgabe:

```
37
```

Platz für Ihre Notizen. Diese werden nicht bewertet. Nur was auf dem Judge eingereicht wird, zählt für diese Aufgabe.

/ 16 P

Theorieaufgabe T1.

Hinweise:

1) In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
2) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

/ 1 P

a) Führen Sie auf dem folgenden Array zwei Iterationen des Sortieralgorithmus *Sortieren durch Einfügen* aus. Das zu sortierende Array ist durch vorherige Iterationen dieses Algorithmus bereits bis zum Doppelstrich sortiert worden.

3	15	20	32	19	5	25	18	21	17	16	45
1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12

/ 1 P

b) Geben Sie an, wie viele Schlüsselvergleiche benötigt werden, wenn unter Verwendung der Move-to-Front Regel auf die folgende Liste eine Abfrage der Elemente T, S, I und L (in dieser Reihenfolge) erfolgt:

$$S \to I \to L \to E \to N \to T$$

Anzahl Vergleiche: _____

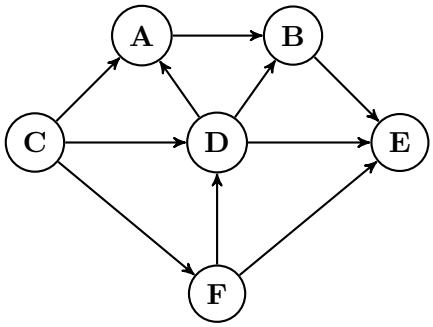
/ 1 P

c) Fügen Sie die Schlüssel $17, 19, 4, 26$ in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie Double Hashing mit der Hashfunktion $h(k) = k \bmod 11$ und benutzen Sie $h'(k) = 1 + (k \mod 9)$ zur Sondierung (*nach links*).

				15		6	18			21
0	1	2	3	4	5	6	7	8	9	10

/ 1 P

d) Geben Sie eine topologische Sortierung des untenstehenden Graphen an.



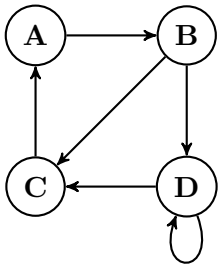
Topologische Sortierung:

_____, _____, _____, _____, _____, _____.

/ 1 P

e) Geben Sie für den folgenden Graphen G die Adjazenzmatrix an. Berechnen Sie seine reflexive, transitive Hülle und geben Sie die entsprechende Adjazenzmatrix an.

Graph G:



Adjazenzmatrix von G:

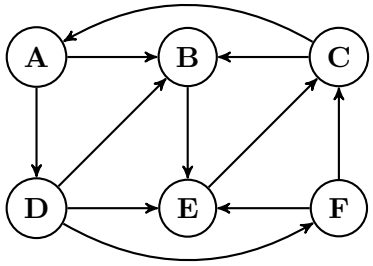
	A	*B*	*C*	*D*
A				
B				
C				
D				

Adjazenzmatrix der reflexiven und transitiven Hülle von G:

	A	*B*	*C*	*D*
A				
B				
C				
D				

/ 1 P

f) Es kann vorkommen, dass die Reihenfolge, in der man die Knoten eines Graphen trifft, für Tiefensuche und Breitensuche dieselbe ist. Markieren Sie im folgenden Graphen alle Startknoten, für die das gilt, unter der Annahme, dass die beiden Traversierungen die jeweiligen Nachbarknoten in alphabetischer Reihenfolge abarbeiten.



/ 1 P

g) Zeichnen Sie denjenigen binären Suchbaum, bei dem in Postorder-Reihenfolge die Schlüssel $1, 4, 3, 10, 7, 6$ angetroffen werden.

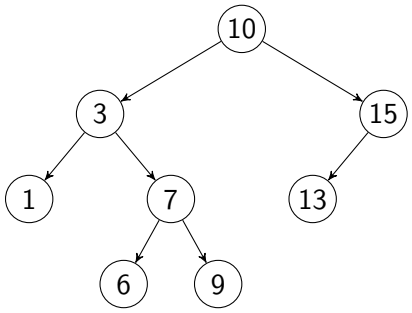
/ 2 P

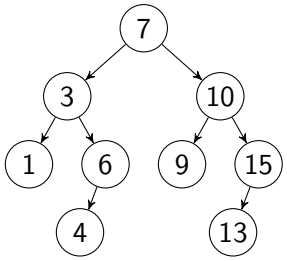
h) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche Antwort werden 0,5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

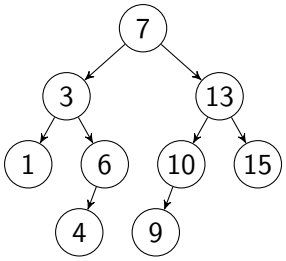
Heapsort ist in-situ und stabil.	☐ Wahr	☐ Falsch
In einem Min-Heap ist das grösste Element in einem Knoten ohne Nachfolger gespeichert.	☐ Wahr	☐ Falsch
Die Höhe eines natürlichen binären Suchbaums ist in $\mathcal{O}(\log n)$, wobei n die Anzahl der im Suchbaum gespeicherten Schlüssel ist.	☐ Wahr	☐ Falsch
Ein gerichteter Graph ist genau dann azyklisch, wenn er eine topologische Sortierung besitzt.	☐ Wahr	☐ Falsch

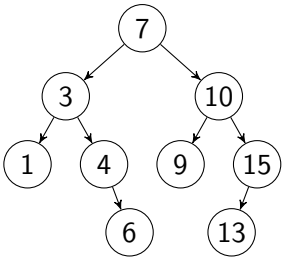
/ 1 P

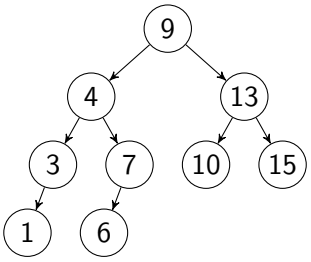
i) Welcher AVL-Baum entsteht, wenn im folgenden AVL-Baum der Schlüssel 4 eingefügt und danach die AVL-Bedingung wiederhergestellt wird?

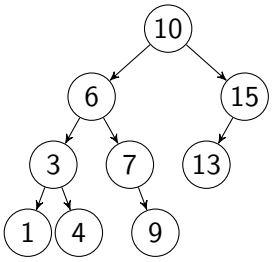


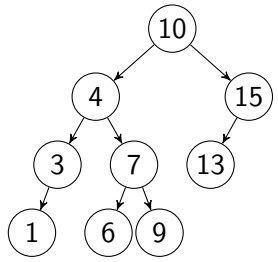
☐ 

☐ 

☐ 

☐ 

☐ 

☐ 

/ 3 P

j) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 5 \cdot T(n/7) + 8 & n > 1 \\ 3 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (nicht-rekursive) und *möglichst einfache* Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion. Sie können annehmen, dass n eine Potenz von 7 ist. Benutzen Sie also $n = 7^k$ oder $k = \log_7(n)$.

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

Herleitung (falls benötigt):

Geschlossene und vereinfachte Formel:

$T(n) = T(7^k) =$

Induktionsbeweis:

Induktionsbeweis (Fortsetzung):

/ 1 P

k) Geben Sie für das folgende Codefragment in Θ-Notation (so knapp wie möglich) an, wie oft die Funktion f in Abhängigkeit von $n \in \mathbb{N}$ asymptotisch gesehen aufgerufen wird. Die Funktion f ruft sich nicht selbst wieder auf. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 1; i <= 2*n; i = i+5 ) {
2         for(int j = 1; j*j <= n; j = j+1 )
3                 f();
4         for(int k = 1; k*k < 1000; k = k+1 )
5                 f();
6 }
```

Anzahl der Funktionsaufrufe in möglichst knapper Θ-Notation:

/ 1 P

l) Geben Sie für das folgende Codefragment in Θ-Notation (so knapp wie möglich) an, wie oft die Funktion f in Abhängigkeit von $n \in \mathbb{N}$ asymptotisch gesehen aufgerufen wird. Die Funktion f ruft sich nicht selbst wieder auf. Sie müssen Ihre Antwort nicht begründen.

```
1 for ( int i = 1; i < n; i = 2*i ) {
2         for ( int j = n; j > 1; j = j/2 )
3                 f();
4 }
```

Anzahl der Funktionsaufrufe in möglichst knapper Θ-Notation:

/ 1 P

m) 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^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^7)$ und $n^7 \in \mathcal{O}(n^9)$ gilt.

$$\binom{n}{3},\ n!,\ 10^8,\ n^{\frac{3}{2}},\ \sqrt{n}\log n,\ \frac{n}{\log^3(n)},\ 2^n,\ 3^{\frac{n}{2}},\ \log(n^6)$$

Lösung: _____, _____, _____, _____, _____, _____, _____, _____, _____.

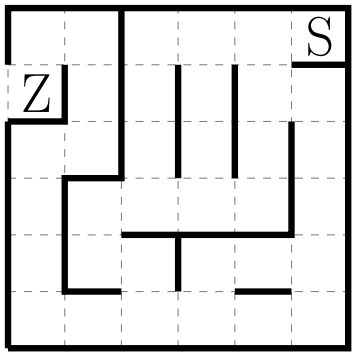
/ 12 P

Theorieaufgabe T2.

König Minos beauftragt Dädalus, ein Labyrinth zu bauen, um dort den Minotaurus einzusperren. Dädalus präsentiert sein Labyrinth mit n Feldern und einem gegebenen Startfeld als Zeichnung auf kariertem Papier. In der folgenden Abbildung ist ein Beispiellabyrinth mit $n = 36$ Feldern gezeichnet. Das Startfeld ist mit S gekennzeichnet, das Zielfeld beim Ausgang mit Z. Wir interessieren uns dafür, wie schnell der Minotaurus aus einem gegebenen Labyrinth entkommen kann.

/ 4 P a) Modellieren Sie dieses Problem als Kürzeste-Wege-Problem:

- Beschreiben Sie, wie man das Labyrinth als Graphen darstellen kann, sodass Folgendes gilt: Die Anzahl Knoten auf einem kürzesten Weg zwischen zwei Knoten, welche das gegebene Start- und Zielfeld repräsentieren, entspricht genau der Anzahl der Felder, die mindestens besucht werden müssen, um das Zielfeld Z zu erreichen.
- Geben Sie an, wie viele Knoten und Kanten Ihr Graph in Abhängigkeit von n hat.
- Nennen Sie einen Algorithmus aus der Vorlesung, der das Kürzeste-Wege-Problem für diesen Graph möglichst effizient löst. Geben Sie auch die Laufzeit so vereinfacht wie möglich in Θ-Notation an.

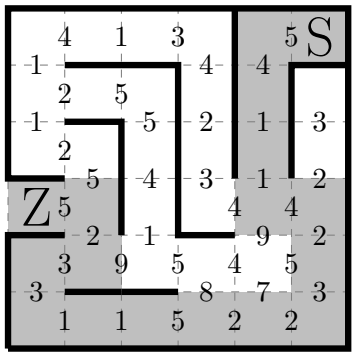


Beispiel: Im Beispiel rechts muss der Minotaurus mindestens 21 Felder (inklusive dem Start- und Zielfeld) besuchen, um zu entkommen.

/ 4 P b) Verschiedene Hindernisse sollen die Flucht erschweren. Die Zeit, die benötigt wird, um von einem Feld ins nächste zu gelangen, ändert sich je nach Hindernis. Für zwei benachbarte Felder, die nicht durch eine Mauer getrennt sind, ist jeweils die Zeit angegeben, um von einem Feld zum anderen zu gelangen.

Wie können Sie Ihre Modellierung aus Aufgabe a) anpassen, um unter Berücksichtigung dieser Zeitangaben den schnellsten Fluchtweg aus dem Labyrinth zu berechnen?

- Beschreiben Sie, wie man das Labyrinth als Graphen darstellen kann, sodass Folgendes gilt: Die Länge eines kürzesten Wegs zwischen zwei Knoten, welche das gegebene Start- und Zielfeld repräsentieren, entspricht genau der mindestens benötigten Zeit, um das Zielfeld Z zu erreichen.
- Geben Sie an, wie viele Knoten und Kanten Ihr Graph in Abhängigkeit von n hat.
- Nennen Sie einen Algorithmus aus der Vorlesung, der das Kürzeste-Wege-Problem für diesen Graph möglichst effizient löst. Geben Sie auch die Laufzeit so vereinfacht wie möglich in Θ-Notation an.

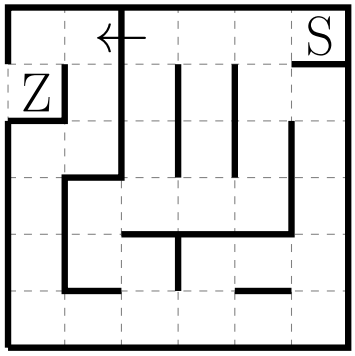


Beispiel: Im Beispiel rechts benötigt man mindestens 44 Zeiteinheiten für die Flucht. Der schnellste Weg ist grau markiert.

/ 4 P

c) Der Minotaurus hat die Kraft, eine einzige Innenwand des Labyrinths (d.h. eine Wand, bei der beide angrenzenden Felder innerhalb des Labyrinths liegen) einzureissen.

Berechnen Sie, wie viele Felder der Minotaurus dann auf seiner Flucht mindestens besuchen muss, indem Sie das Problem wiederum als Kürzeste-Wege-Problem modellieren.
Nennen Sie einen Algorithmus, der dieses Problem möglichst effizient löst. Geben Sie auch die Laufzeit so vereinfacht wie möglich in Θ-Notation an.
Beispiel: Im Beispiel rechts kann der Minotaurus die mit dem Pfeil markierte Mauer einreissen und muss so nur noch 7 Felder besuchen (und nicht mehr 21 wie in Aufgabe a)).



Teilaufgabe a)

- Definition des Graphen (wenn möglich in Worten und nicht formal):

- Anzahl der Knoten und Kanten (in möglichst knapper Θ-Notation):

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs:

- Laufzeit (in möglichst knapper Θ-Notation):

Teilaufgabe b)

- Definition des Graphen (wenn möglich in Worten und nicht formal):

- Anzahl der Knoten und Kanten (in möglichst knapper Θ-Notation):

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs:

- Laufzeit (in möglichst knapper Θ-Notation):

Teilaufgabe c)

- Modellierung als Kürzeste-Wege-Problem:

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs:

- Laufzeit (in möglichst knapper Θ-Notation):

/ 12 P

Theorieaufgabe T3.

Das längste Baguette der Welt hat eine Länge von l Zentimetern. Es soll so teuer wie möglich verkauft werden. Es darf dafür in Stücke geschnitten werden. Jedes Stück der Länge $l_i \in \mathbb{N}$ kann zu einem Preis $p_i \in \mathbb{N}$ verkauft werden, für $i \in \{1, \ldots, n\}$. Wir wollen ermitteln, wie das Baguette so in Stücke geschnitten werden kann, dass der höchstmögliche Gesamtverkaufspreis erzielt wird. Natürlich darf dabei die Gesamtlänge des Baguettes durch die Summe der Längen der Stücke nicht überschritten werden.

Beispiel: Gegeben sei ein Baguette der Länge $l = 121$ cm, und wir erlauben $n = 3$ verschiedene Stücklängen, nämlich $l_1 = 50$ cm (mit Preis $p_1 = 25$), $l_2 = 26$ cm (mit Preis $p_2 = 14$) sowie $l_3 = 20$ cm (mit Preis $p_3 = 10$). Dann verkaufen wir am besten drei Stücke der Länge 26 cm und zwei der Länge 20 cm, zum Gesamtpreis von $3 \cdot 14 + 2 \cdot 10 = 62$.

/ 7 P a) Geben Sie einen Algorithmus nach dem Prinzip der dynamischen Programmierung an, der als Eingabe l und n sowie l_i und p_i für alle $i \in \{1, \ldots, n\}$ erhält, und der den höchstmöglichen Gesamtverkaufspreis berechnet. Für das vorige Beispiel soll also die Zahl 62 berechnet werden. Erklären Sie dabei die folgenden Aspekte:

1) Was ist die Bedeutung eines Tabelleneintrags, und welche Grösse hat die DP-Tabelle?
2) Wie kann die Tabelle initialisiert werden, und wie berechnet sich ein Tabelleneintrag aus früher berechneten Einträgen?
3) In welcher Reihenfolge können die Einträge berechnet werden?
4) Wie lässt sich die Lösung aus der DP-Tabelle auslesen?

/ 3 P b) Beschreiben Sie, wie man ermitteln kann, welche Stückelung zum höchstmöglichen Gesamtverkaufspreis führt. Für das vorige Beispiel soll also bestimmt werden, dass die beste Stückelung aus drei Stücken der Länge 26 cm zum Preis von je 14 und aus zwei Stücken der Länge 20 cm zum Preis 10 besteht.

/ 2 P c) Geben Sie die asymptotischen Laufzeiten Ihrer Algorithmen für die Aufgabenteile a) und b) an und begründen Sie diese.

Teilaufgabe a)

Grösse der DP-Tabelle / Anzahl Einträge: _____

Bedeutung eines Tabelleneintrags:

$DP[\_\_\_\_\_]$: _____

_____

Berechnung eines Eintrags:

Berechnungsreihenfolge:

Berechnung des höchstmöglichen Gesamtverkaufspreises:

Teilaufgabe b)

Ermittlung der optimalen Stückelung:

Teilaufgabe c)

Laufzeiten (in möglichst knapper Θ-Notation) mit Begründung:

Extraplatz. Bitte kennzeichnen Sie deutlich zu welcher Aufgabe Ihre Aufschrift gehört. Notizen, die Sie nicht bewerten lassen möchten, bitte durchstreichen.