**/ 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 |

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

| 3 | 5 | 15 | 19 | 20 | 32 ‖ | 25 | 18 | 21 | 17 | 16 | 45 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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: 15

1) Abfrage T: 6 Vergleiche $T \to S \to I \to L \to E \to N$
2) Abfrage S: 2 Vergleiche $S \to T \to I \to L \to E \to N$
3) Abfrage I: 3 Vergleiche $I \to S \to T \to L \to E \to N$
4) Abfrage L: 4 Vergleiche $L \to I \to S \to T \to E \to N$

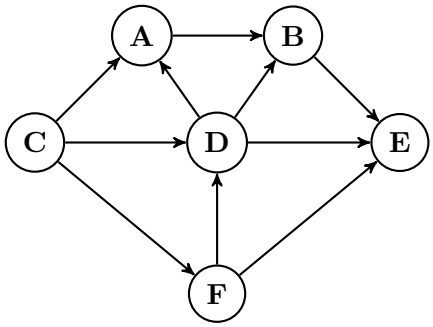
**/ 1 P**

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*).

| | 26 | 19 | | 15 | 4 | 6 | 18 | 17 | | 21 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

**/ 1 P**

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



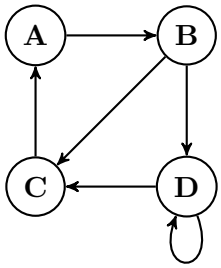

Topologische Sortierung:

$C, F, D, A, B, E$

**/ 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*:

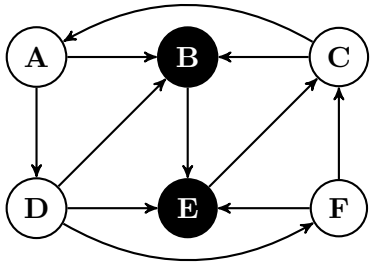
| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | 1 | 0 | 0 |
| $B$ | 0 | 0 | 1 | 1 |
| $C$ | 1 | 0 | 0 | 0 |
| $D$ | 0 | 0 | 1 | 1 |

*Adjazenzmatrix der transitiven und reflexiven Hülle von G*:

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 1 | 1 | 1 | 1 |
| $B$ | 1 | 1 | 1 | 1 |
| $C$ | 1 | 1 | 1 | 1 |
| $D$ | 1 | 1 | 1 | 1 |

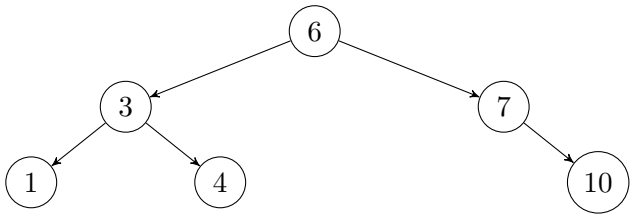
**/ 1 P**

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.



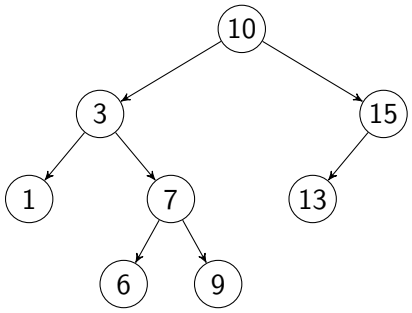

**/ 2 P**

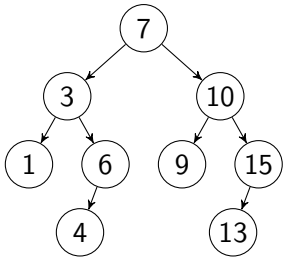
h) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche Antwort werden 0,5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

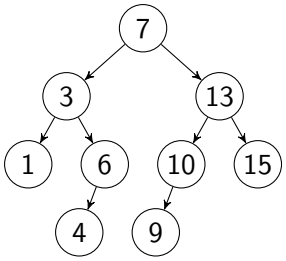
| | | |
|---|---|---|
| *Heapsort ist in-situ und stabil.* | ☐ Wahr | ☒ Falsch |
| *In einem Min-Heap ist das grösste Element in einem Knoten ohne Nachfolger gespeichert.* | ☒ Wahr | ☐ Falsch |
| *Die Höhe eines natürlichen binären Suchbaums ist in $\mathcal{O}(\log n)$, wobei $n$ die Anzahl der im Suchbaum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

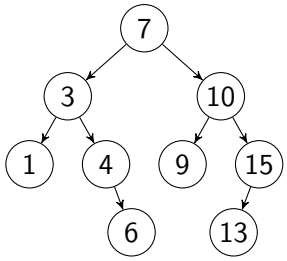
**/ 1 P**

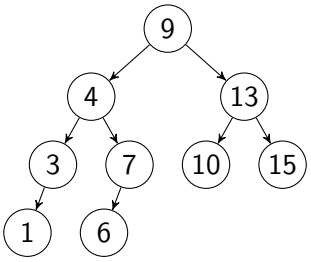
i) Welcher AVL-Baum entsteht, wenn im folgenden AVL-Baum der Schlüssel 4 eingefügt und danach die AVL-Bedingung wiederhergestellt wird?

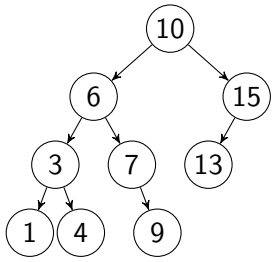


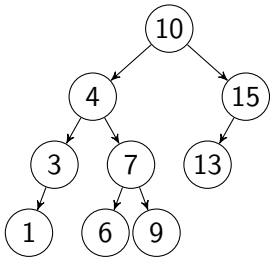

☐



☐



☐



☐



■



☐


**/ 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):*

Da wir annehmen dürfen, dass $n$ eine Potenz von 7 ist, gilt $n = 7^k$ für ein $k \in \mathbb{N}$. Wir teleskopieren, um auf eine Formel für $T(n)$ zu kommen:

$$\begin{aligned} T(n) &= 5 \cdot T(n/7) + 8 \\ &= 5 \cdot \big(5 \cdot T(n/7^2) + 8\big) + 8 \\ &= 5 \cdot \Big(5 \cdot \big(5 \cdot T(n/7^3) + 8\big) + 8\Big) + 8 = ... \\ &= 5^k \cdot 3 + 8 \cdot \sum_{i=0}^{k-1} 5^i \\ &= 5^k \cdot 3 + 8 \cdot \frac{5^k - 1}{4} \\ &= 5^k \cdot 3 + 2 \cdot 5^k - 2 \\ &= 5^{k+1} - 2 \end{aligned}$$

*Geschlossene und vereinfachte Formel:*

$$T(n) = T(7^k) = 5^{k+1} - 2$$

*Induktionsbeweis:*

Wir beweisen nun unsere Annahme durch vollständige Induktion über $k$.

*Induktionsverankerung* ($k = 0$): Es gilt $T(7^0) = T(1) = 3 = 5^1 - 2$.

*Induktionsannahme*: Für ein $k \in \mathbb{N}_0$ sei $T(7^k) = 5^{k+1} - 2$.

*Induktionsschritt* ($k \to k + 1$):

$$T(7^{k+1}) = 5 \cdot T(7^k) + 8 \overset{\text{Ind.-Ann.}}{=} 5 \cdot (5^{k+1} - 2) + 8 = 5 \cdot 5^{k+1} - 10 + 8 = 5^{k+2} - 2.$$

*Induktionsbeweis (Fortsetzung):*

—

**/ 1 P**

k) Geben Sie für das folgende Codefragment in $\Theta$-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 $\Theta$-Notation:*

$$\Theta(n^{3/2})$$

*Lösung:* Die äussere Schleife wird $\mathcal{O}(n)$ Mal durchlaufen. Die erste innere Schleife inkrementiert den Wert von $j$ in jedem Schritt um 1 bis $j^2 > n$. Daher wird sie $\Theta(\sqrt{n})$ Mal durchlaufen. Die zweite innere Schleife wird nur konstakt oft durchlaufen. Also wird $f$ asymptotisch $\Theta(n \cdot (1 + \sqrt{n})) = \Theta(n\sqrt{n})$ Mal aufgerufen.

**/ 1 P**

l) Geben Sie für das folgende Codefragment in $\Theta$-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 $\Theta$-Notation:*

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

*Lösung:* Sowohl die äussere als auch die innere Schleife werden $\Theta(\log n)$ Mal durchlaufen. Daher liegt die Anzahl der Aufrufe von $f$ in $\Theta(\log^2(n))$.

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

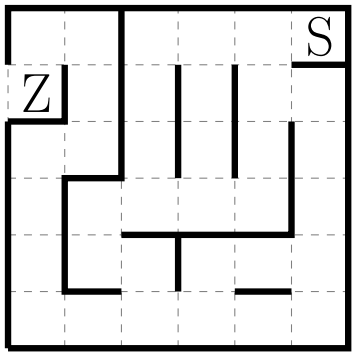
**/ 12 P**

**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 $\Theta$-Notation an.

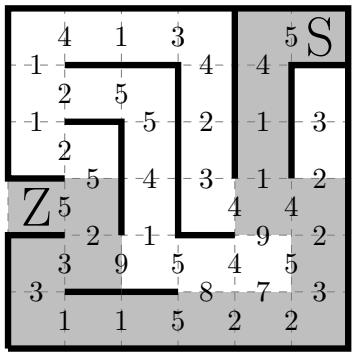



*Beispiel:* Im Beispiel rechts muss der Minotaurus mindestens 21 Felder (inklusive dem Start- und Zielfeld) besuchen, um zu entkommen.

**/ 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 $\Theta$-Notation an.

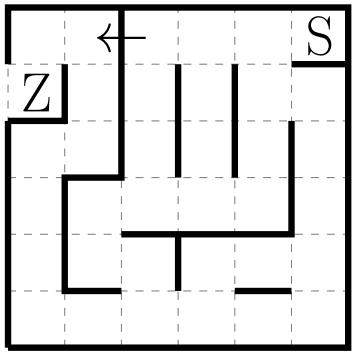



*Beispiel:* Im Beispiel rechts benötigt man mindestens 44 Zeiteinheiten für die Flucht. Der schnellste Weg ist grau markiert.

**/ 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 $\Theta$-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):

  *Wir erzeugen einen ungerichteten Graphen, der genau einen Knoten für jedes Feld des Labyrinths enthält. Zwei Knoten werden durch eine Kante miteinander verbunden, wenn die entsprechenden Felder benachbart und durch keine Mauer voneinander getrennt sind.*

- Anzahl der Knoten und Kanten (in möglichst knapper $\Theta$-Notation):

  *Das Labyrinth hat $n$ Felder, also hat der zugehörige Graph $n \in \Theta(n)$ Knoten. Jeder Knoten ist zu maximal vier Kanten inzident, also hat der gesamte Graph auch höchstens $\Theta(n)$ Kanten.*

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs:

  *Eine Breitensuche kann benutzt werden, um einen kürzesten Weg von dem Knoten, der $S$ repräsentiert zu dem Knoten, der $Z$ repräsentiert, zu berechnen.*

- Laufzeit (in möglichst knapper $\Theta$-Notation):

  *Da der Graph $\Theta(n)$ viele Knoten und auch $\Theta(n)$ viele Kanten hat, läuft eine Breitensuche in Zeit $\Theta(|V| + |E|) = \Theta(n + n) = \Theta(n)$.*

## *Teilaufgabe b)*

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

  *Wir erzeugen wieder einen ungerichteten Graphen, der genau einen Knoten für jedes Feld des Labyrinths enthält. Zwei Knoten werden durch eine Kante miteinander verbunden, wenn die entsprechenden Felder benachbart und durch keine Mauer voneinander getrennt sind. Die Kosten der Kante entsprechen genau den Kosten, um vom einen Feld zum anderen zu gelangen.*

- Anzahl der Knoten und Kanten (in möglichst knapper $\Theta$-Notation):

  *Das Labyrinth hat $n$ Felder, also hat der zugehörige Graph $n \in \Theta(n)$ Knoten. Jeder Knoten ist zu maximal vier Kanten inzident, also hat der gesamte Graph auch höchstens $\Theta(n)$ Kanten.*

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs:

  *Da jetzt alle Kanten des Graphs nicht-negative Gewichte haben und ein Weg mit kleinstem Gesamtgewicht gesucht wird, kann Dijkstras Algorithmus verwendet werden.*

- Laufzeit (in möglichst knapper $\Theta$-Notation):

  *Der Graph hat $|V| \in \Theta(n)$ viele Knoten und $|E| \in \Theta(n)$ viele Kanten. Dijkstras Algorithmus hat, wenn er mit einem Fibonacci-Heap implementiert wird, eine Laufzeit von $\mathcal{O}(|E| + |V| \log |V|) = \mathcal{O}(n + n \log n) = \mathcal{O}(n \log n)$. Wird Dijkstras Algorithmus mit einem herkömmlichen Heap implementiert, beträgt die Laufzeit $\mathcal{O}((|E|+|V|) \log |V|)$, was ebenfalls zu $\mathcal{O}(n \log n)$ vereinfacht werden kann.*

## *Teilaufgabe c)*

- Modellierung als Kürzeste-Wege-Problem:

  *Wir erzeugen zunächst zwei unabhängige, identische Kopien $K_1$ und $K_2$ des in a) entwickelten Graphen. In beiden Graphen werden alle ungerichteten Kanten durch je zwei gerichtete Kanten, eine für jede Richtung, ersetzt. Die erste Kopie, $K_1$ repräsentiert die Situation in der noch keine Mauer durchbrochen wurde, und $K_2$ repräsentiert die Situation, in der bereits eine Mauer durchbrochen wurde.*

  *Wir gehen nun über alle Paare zweier benachbarter Felder des Labyrinths, die durch eine Mauer voneinander getrennt sind. Seien $v_1$ und $v_2$ die Knoten eines der beiden Felder in $K_1$ bzw. in $K_2$, und seien $w_1$ und $w_2$ die Knoten des anderen Feldes in $K_1$ bzw. in $K_2$. Wir erzeugen nun eine gerichtete Kante von $v_1$ nach $w_2$, und eine weitere gerichtete Kante von $w_1$ zu $v_2$, um den Durchbruch der entsprechenden Mauer zu repräsentieren.*

  *Gesucht wird jetzt ein kürzester Weg von $S_1$, dem Knoten in $K_1$, der das Startfeld $S$ repräsentiert, zu $Z_1$ oder $Z_2$ (den Knoten des Zielfeldes in $K_1$ bzw. $K_2$). Sobald entweder $Z_1$ oder $Z_2$ gefunden wird, ist die Suche beendet.*

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs:

  *Eine Breitensuche kann benutzt werden, um einen kürzesten Weg von dem Knoten in $K_1$, der $S$ repräsentiert zu einem der beiden Knoten, die $Z$ in $K_1$ oder in $K_2$ repräsentieren, zu berechnen.*

- Laufzeit (in möglichst knapper $\Theta$-Notation):

  *Da der Graph $\Theta(n)$ viele Knoten und auch $\Theta(n)$ viele Kanten hat, läuft eine Breitensuche in Zeit $\Theta(|V| + |E|) = \Theta(n + n) = \Theta(n)$.*

**/ 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:**

*Wir benutzen eine Tabelle $DP$ der Grösse $1 \times (l+1)$, deren Einträge von $0$ bis $l$ indiziert sind.*

**Bedeutung eines Tabelleneintrags:**

*$DP[g]$ gibt den höchstmöglichen Gesamtverkaufspreis an, wenn nur Stücke verwendet werden, deren Gesamtlänge $g$ nicht überschreitet.*

**Berechnung eines Eintrags:**

*Zur Initialisierung setzen wir $DP[0] = 0$, denn unter Verwendung von Stücken der Länge 0 kann kein Gewinn erzielt werden. Für $g \in \{1, \ldots, l\}$ setzen wir*

$$DP[g] = \max \Big\{ DP[g - l_i] + p_i \;\Big|\; i \in \{1, \ldots, n\} \text{ und } l_i \leq g \Big\},$$

*denn unter Verwendung des $i$-ten Stücks haben wir den Preis $p_i$ erzielt, und müssen mit Stücken der Gesamtlänge von höchstens $g - l_i$ einen höchstmöglichen Gesamtverkaufspreis erzielen. Da die beste Wahl von $i$ im Vorfeld nicht bekannt ist, muss das Maximum über alle möglichen Wahlen für $i$ gebildet werden. Man beachte, dass $g - l_i$ niemals negativ werden darf, weswegen das Maximum nur über die Indizes $i$ mit $l_i \leq g$ gebildet wird.*

**Berechnungsreihenfolge:**

*Wir berechnen die Einträge $DP[g]$ nach aufsteigendem $g$, also zuerst $DP[0]$, dann $DP[1]$, dann $DP[2]$, usw.*

**Berechnung des höchstmöglichen Gesamtverkaufspreises:**

*Der höchstmögliche Gesamtverkaufspreis ist nach Abschluss der Berechnung in $DP[l]$ gespeichert.*

## *Teilaufgabe b)*

**Ermittlung der optimalen Stückelung:**

*Wir setzen zunächst $g \leftarrow l$. Danach rekonstruieren wir schrittweise eine Stückelung, die zum höchstmöglichen Gesamtverkaufspreis führt. Dazu berechnen wir zunächst ein $i$ mit $l_i \leq g$ und $DP[g] = DP[g - l_i] + p_i$. Wir geben dann die Länge $l_i$ aus, setzen $g \leftarrow g - l_i$ und fahren auf die gleiche Art fort. Falls es keinen solchen Index $i$ mit $l_i \leq g$ und $DP[g] = DP[g - l_i] + p_i$ mehr gibt, wurden alle Stücke rekonstruiert und wir sind fertig.*

## *Teilaufgabe c)*

**Laufzeiten (in möglichst knapper $\Theta$-Notation) mit Begründung:**

*Die DP-Tabelle hat $\Theta(l)$ viele Einträge, und zur Berechnung eines Eintrags müssen im schlechtesten Fall $n$ andere Einträge der Tabelle angeschaut werden. Dieser Fall kann auch tatsächlich eintreten, wenn die Stücklängen $l_i$ relativ zur Gesamtlänge $l$ klein sind. Die Laufzeit zur Berechnung der Tabelle beträgt im schlechtesten Fall also $\Theta(nl)$.*

*Bei der Rekonstruktion ist der Wert von $g$ initial $l$, und dieser Wert wird im schlechtesten Fall in jedem Schritt nur um 1 verringert. Damit werden in der Rekonstruktion bis zu $l$ Schritte gemacht. In jedem Schritt werden bis zu $n$ Tabelleneinträge mit $DP[g]$ verglichen, was Zeit $\Theta(n)$ kostet. Damit liegt die Laufzeit zur Rekonstruktion der Tabelle im schlechtesten Fall ebenfalls in $\Theta(nl)$.*