Lehrstuhl für Informatikanwendungen in der Medizin
Fakultät für Informatik
Technische Universität München

Hinweise zur Personalisierung:
- Ihre Prüfung wird bei der Anwesenheitskontrolle durch Aufkleben eines Codes personalisiert.
- Dieser enthält lediglich eine fortlaufende Nummer, welche auch auf der Anwesenheitsliste neben dem Unterschriftenfeld vermerkt ist.
- Diese wird als Pseudonym verwendet, um eine eindeutige Zuordnung Ihrer Prüfung zu ermöglichen.

Grundlagen Algorithmen und Datenstrukturen

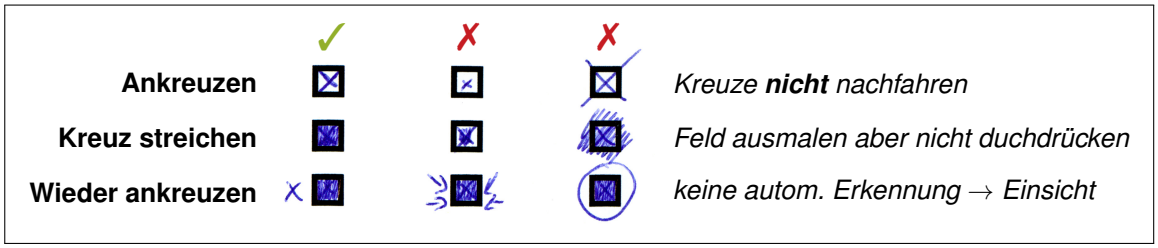
Klausur: IN0007 / Endterm **Datum:** Samstag, 29. Juli 2017
Prüfer: PD Dr. Tobias Lasser **Uhrzeit:** 10:30 – 13:00

	A 1	A 2	A 3	A 4	A 5	A 6	A 7	A 8
I								
II								

Bearbeitungshinweise

- Diese Klausur umfasst
  - **24 Seiten** mit insgesamt **8 Aufgaben**

  Bitte kontrollieren Sie jetzt, dass Sie eine vollständige Angabe erhalten haben.
- Das Heraustrennen von Seiten aus der Prüfung ist untersagt.
- Mit * gekennzeichnete Teilaufgaben sind ohne Kenntnis der Ergebnisse vorheriger Teilaufgaben lösbar.
- **Es werden nur solche Ergebnisse gewertet, bei denen der Lösungsweg erkennbar ist.** Auch Textaufgaben sind **grundsätzlich zu begründen**, sofern es in der jeweiligen Teilaufgabe nicht ausdrücklich anders vermerkt ist.
- Schreiben Sie weder mit roter / grüner Farbe noch mit Bleistift.
- Die Gesamtpunktzahl in dieser Prüfung beträgt 150 Punkte.
- Als Hilfsmittel sind zugelassen:
  - ein **selbst-beschriebenes Hilfsblatt** (A4 double-sided nicht kopiert oder bedruckt)
  - ein **analoges Wörterbuch** Deutsch ↔ Muttersprache **ohne Anmerkungen**
- Schalten Sie alle mitgeführten elektronischen Geräte vollständig aus, verstauen Sie diese in Ihrer Tasche und verschließen Sie diese.



Aufgabe 1 **Wissensfragen (8 Punkte)**

Bei jeder der folgenden Teilaufgaben ist genau eine Aussage richtig. Kreuzen Sie jeweils nur die richtige Aussage an. Alle Teilfragen lassen sich unabhängig voneinander beantworten.

a)* Welche Aussage über *ungerichtete Graphen* ist korrekt?

- ☐ Der Algorithmus von Dijkstra kann bei beliebigen Kantengewichten den kürzesten Pfad zwischen zwei Knoten bestimmen.
- ☐ Wenn alle Kantengewichte gleich 2 sind, ist der Graph *2-fach zusammenhängend*.
- ☒ Zusammenhangskomponenten können mit (evtl. modifizierter) Tiefen- oder Breitensuche in $\mathcal{O}(n+m)$ bestimmt werden.
- ☐ Artikulationsknoten (cut-vertices) werden eingesetzt, um das Löschen von Knoten im Graphen effizienter zu gestalten.

b)* Was ist der *eigentliche Rand* des Wortes "baadacbbcabaad"?

- ☒ baad
- ☐ ϵ
- ☐ bd
- ☐ baadacb

c)* Was ist die *Suchbaum-Invariante* für alle Arten von Suchbäumen?

- ☐ Jeder Knoten hat einen kleineren Schlüssel als seine Kinder.
- ☒ Für die Schlüssel l im linken und r im rechten Teilbaum eines Knotens mit Schlüssel x gilt $l \leq x < r$.
- ☐ Alle Blätter befinden sich auf derselben Tiefe.
- ☐ Alle inneren Knoten haben denselben Grad.

d)* Welche Aussage über *RadixSort* ist richtig?

- ☐ Er sortiert auch im Worst-Case in $\mathcal{O}(n \log n)$, ist aber nicht stabil.
- ☐ Er kann nur auf Zahlen im Dezimalsystem angewandt werden.
- ☐ Im Worst-Case ist er nicht schneller als BubbleSort.
- ☒ Er kann richtig implementiert auch im Average-Case schneller sortieren als $\mathcal{O}(n \log n)$.

e)* Welche Eigenschaft gilt für *dynamische Arrays*?

- ☐ Elemente lassen sich schnell in der Mitte des Arrays einfügen oder löschen.
- ☒ Dynamische Arrays sind günstiger für das Caching des Prozessors, da alle Elemente direkt hintereinander im Speicher liegen.
- ☐ Dynamische Arrays lassen sich schneller sortieren als statische.
- ☐ Wenn die ursprünglich erwartete Anzahl an Elementen in das Array eingefügt wurde, können keine weiteren Elemente mehr hinzugefügt werden.

f)* Was lässt sich über die Verschmelzung von zwei *Binomial Heaps* sagen?

- ☐ Der Min-Zeiger muss nach der Verschmelzung auf den Baum mit dem kleinsten Rang zeigen.
- ☐ Nach der Verschmelzung darf man nur einen einzigen Binomial-Baum haben.
- ☐ Sie läuft im Worst-Case in $\Omega(n)$.
- ☒ Sie lässt sich so ähnlich implementieren, wie die bitweise Addition zweier Binärzahlen.

g)* Beim Hashing mit *Linear Probing*...

- ☐ müssen alle Elemente zu Beginn auf einmal gespeichert werden, da die Tabelle später nicht mehr verändert werden kann.
- ☒ ist das Löschen von Elementen schwierig, da Löcher in der Hashtabelle das Auffinden von anderen Elementen verhindern können.
- ☐ ist die Hashfunktion nicht besonders wichtig, da auf ineffiziente Listen verzichtet wird.
- ☐ werden Elemente, deren Schlüssel auf den gleichen Wert gehasht werden, in einer Liste abgelegt.

h)* In welcher Wachstumsordnung liegt die Worst-Case-Laufzeit jedes *vergleichsbasierten Sortieralgorithmus*?

- ☐ $\mathcal{O}(n^2)$
- ☐ $o(n \log n)$
- ☒ $\Omega(n \log n)$
- ☐ $\mathcal{O}(k + d)$

Aufgabe 2 MergeSort (11 Punkte)

0 1 2 3 4 5 6 7 8

a) Sortieren Sie die folgende Zahlenfolge mit MergeSort:

$$21, 82, 2, 34, 79, 9, 65, 60, 95, 38, 62, 88, 90, 7, 94, 49$$

Geben Sie für jede Rekursionsebene jeweils für das Aufspalten der Teilsequenzen und für das Verschmelzen der sortierten Teilsequenzen einen Zwischenschritt an (d.h. bei dieser Eingabesequenz insgesamt circa acht Zwischenschritte), sodass Ihr Vorgehen nachvollzogen werden kann.

Aufspalten: 21, 82, 2, 34, 79, 9, 65, 60 |||| 95, 38, 62, 88, 90, 7, 94, 49
Aufspalten: 21, 82, 2, 34 ||| 79, 9, 65, 60 |||| 95, 38, 62, 88 ||| 90, 7, 94, 49

Aufspalten: 21, 82 || 2, 34 ||| 79, 9 || 65, 60 |||| 95, 38 || 62, 88 ||| 90, 7 || 94, 49

Aufspalten: 21 | 82 || 2 | 34 ||| 79 | 9 || 65 | 60 |||| 95 | 38 || 62 | 88 ||| 90 | 7 || 94 | 49

Verschmelzen: 21, 82 || 2, 34 ||| 9, 79 || 60, 65 |||| 38, 95 || 62, 88 ||| 7, 90 || 49, 94

Verschmelzen: 2, 21, 34, 82 ||| 9, 60, 65, 79 |||| 38, 62, 88, 95 ||| 7, 49, 90, 94

Verschmelzen: 2, 9, 21, 34, 60, 65, 79, 82 |||| 7, 38, 49, 62, 88, 90, 94, 95

Verschmelzen: 2, 7, 9, 21, 34, 38, 49, 60, 62, 65, 79, 82, 88, 90, 94, 95

0 1 2 3

b)* Die meisten Sortieralgorithmen gehen vom theoretischen Idealfall aus, dass die ganze Liste (und jeder benötigte Zusatzspeicher) komplett in den Arbeitsspeicher passt. Wenn dies nicht der Fall ist, da die zu sortierende Liste zu groß ist, um ganz in den Arbeitsspeicher geladen zu werden, kann man das Prinzip des *externen Sortierens* nutzen. Dabei wird die Liste im ersten Schritt in Blöcke aufgeteilt, die komplett im Speicher sortiert werden können. Im zweiten Schritt werden diese sortierten Blöcke in eine sortierte Gesamtliste zusammengefügt.
Geben Sie bitte an, wie gut *MergeSort* für diese Schritte jeweils geeignet ist, und wo anwendbar, welche Teile dafür von MergeSort benötigt werden. Begründen Sie Ihre Antworten jeweils kurz.

Für den ersten Schritt, das blockweise Sortieren, ist MergeSort ungeeignet , da es $\mathcal{O}(n)$ zusätzlichen Speicher zum Sortieren benötigt.

Der zweite Schritt, das Verschmelzen der sortierten Blöcke, entspricht genau der zweiten Phase von MergeSort, daher kann man diesen Teil von MergeSort auch hierfür verwenden.

Aufgabe 3 Amortisierte Laufzeit (20 Punkte)

Gegeben sei folgender Pseudocode, der über die Funktionen `plus(Integer)` und `times(Integer)` Elemente einer Rechnung entgegen nimmt, und das Ergebnis dieser mit der Funktion `equals()` ausgibt. Dabei können Ausdrücke in beliebiger Reihenfolge eingegeben werden, die Berechnung berücksichtigt den Operatorvorrang (also alle Multiplikationen vor allen Additionen). Sie dürfen davon ausgehen, dass keine Eingaben eine ungültige Operation provozieren. Insbesondere wird als erste Funktion immer `plus()`, und als letzte immer `equals()` aufgerufen.

```
Stack operations;

void plus(Integer i) {
   if(operations not empty) {
      // Alle Multiplikationen bisher aufuehren:
      Operation last = operations.popBack();
      while(last.Operator == "times") {
         Operation prev = operations.popBack();
         last.Number = last.Number * prev.Number;
         last.Operator = prev.Operator;
      }
      operations.pushBack(last);
   }

   operations.pushBack(new Operation(i, "plus"));
}

void times(Integer i) {
   operations.pushBack(new Operation(i, "times"));
}

Integer equals() {
   plus(0); // Sicherstellen, dass alle Multiplikationen abgearbeitet sind

   Integer Result = 0;
   while(operations not empty) {
      Operation last = operations.popBack();
      Result = Result + last.Number;
   }

   return Result;
}
```

a) Bestimmen Sie zunächst die tatsächlichen Laufzeiten der einzelnen Funktionen `plus()`, `times()` und `equals()` in Abhängigkeit voneinander und unter der Annahme, dass alle arithmetischen Operationen auf Zahlen und alle Stack-Operationen konstante Laufzeit haben.

0 1 2 3 4 5

$T(plus) = \mathcal{O}(1) + m \cdot \mathcal{O}(1) = \mathcal{O}(1) + m$, wobei m = Anzahl der Multiplikationen

$T(times) = \mathcal{O}(1)$

$T(equals) = T(plus) + n \cdot \mathcal{O}(1) = T(plus) + n$, wobei n = Anzahl der Elemente im Stack

0 1 2 3 4 5

b) Geben Sie nun ein geeignetes Schema $\Delta(\sigma)$ an, um jeder Funktion $\sigma \in \{plus, times, equals\}$ Tokenoperationen im Sinne der Bankkontomethode zuzuweisen.

$\Delta(plus) = 1 - m$, m = Anzahl der Multiplikationen
$\Delta(times) = 1$

$\Delta(equals) = -n$, n = Anzahl der Elemente im Stack

0 1 2 3 4 5

c) Berechnen Sie abschließend die amortisierte Laufzeit $A(\sigma_1, ..., \sigma_n)$ für eine beliebige gültige Folge von n Aufrufen dieser Funktionen. Geben Sie dazu auch die jeweiligen amortisierten Laufzeiten der einzelnen Funktionen $A(\sigma_i)$ an. *Hinweis:* Eine gültige Folge von Operationen fängt immer mit einem Aufruf von `plus()` an, und ruft `equals()` nur genau als letztes auf.

$A(plus) = T(plus) + \Delta(plus) = \mathcal{O}(1) + m + 1 - m = \mathcal{O}(1) + 1 = \mathcal{O}(1)$
$A(times) = T(times) + \Delta(times) = \mathcal{O}(1) + 1 = \mathcal{O}(1)$

$A(equals) = T(equals) + \Delta(equals) = A(plus) + n - n = \mathcal{O}(1)$

$A(\sigma_1, ..., \sigma_n) = \sum_{i=1}^{n}(A(\sigma_i)) = \sum_{i=1}^{n} \mathcal{O}(1) = \mathcal{O}(n)$

0 1 2 3 4 5

d) Begründen Sie, warum Ihr Amortisierungsschema gültig ist.

Das Schema ist gültig, da das Tokenkonto nie negativ wird.
Das Tokenkonto entspricht immer der Anzahl der Elemente auf dem Stack. Bei jeder Einfügeoperation (`times()` und letzte Zeile von `plus()`) wird ein Token eingezahlt , und jedes Mal, wenn Elemente aus dem Stack entfernt werden (`equals()` und Schleife in `plus()`), werden genausoviele Token aus dem Konto bezahlt.

Aufgabe 4 Binärer Heap (22 Punkte)

Gegeben seien folgende Arrays:

Feld A:	2	4	6	5	12	8	11	7	10	18
Feld B:	1	3	4	5	7	9	12	14	22	2

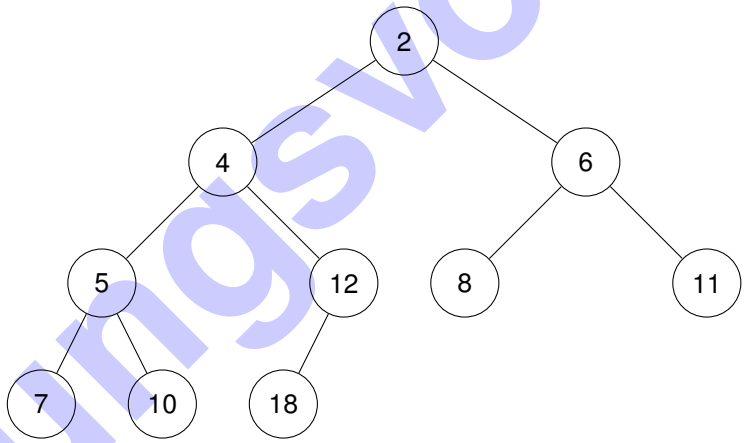
a) Welches von beiden Arrays repräsentiert einen korrekten, binären Min-Heap? Begründen Sie Ihre Antwort kurz.

0
1

Feld A
Das letzte Element in Feld B ist kleiner als die vorhergehenden Elemente im Heap, und verletzt damit die Heap-Invariante.

b) Zeichnen Sie hier eine Baumdarstellung zu dem Array, das einen korrekten Binären Min-Heap darstellt.

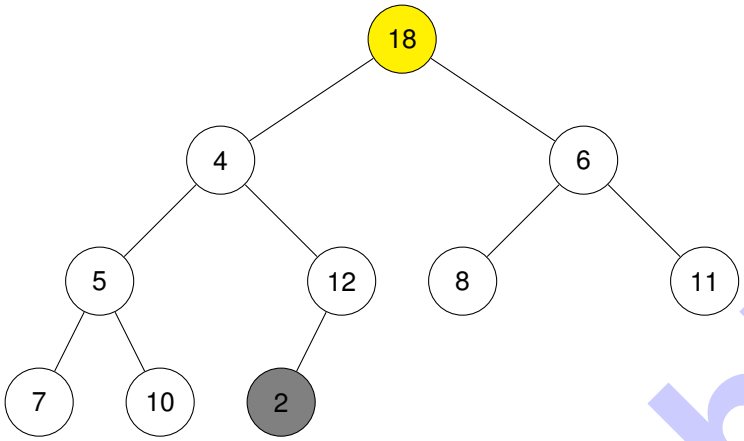
0
1
2



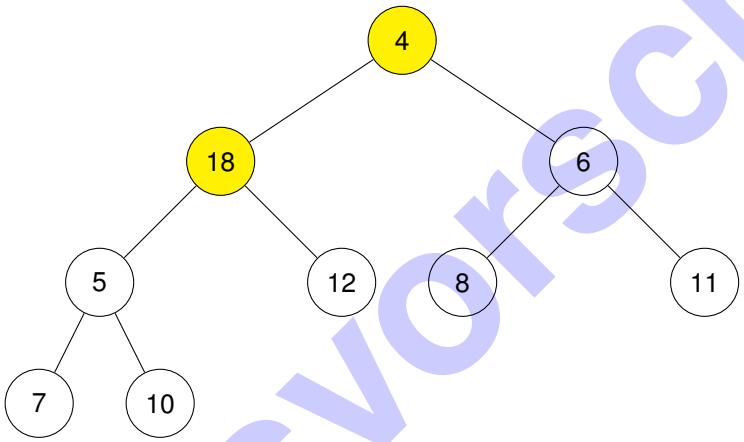
0
1
2
3
4
5
6

c) Führen Sie nun auf diesem Heap die Operation *deleteMin()* aus. Geben Sie dabei alle nötigen Zwischenschritte an.

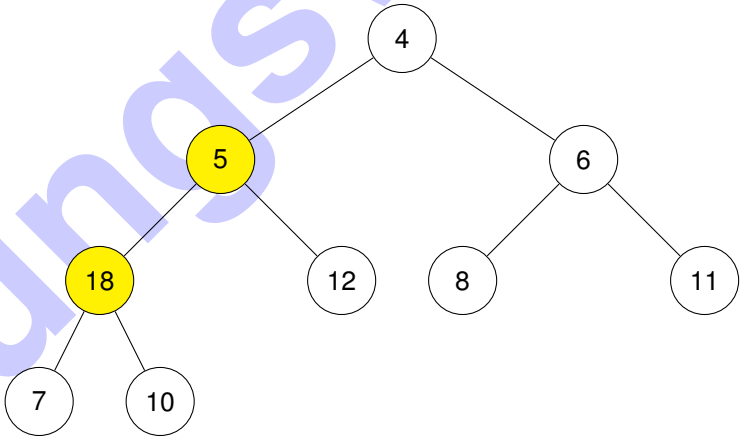
Wurzel/Minimum mit letztem Element vertauschen, letztes Element entfernen :



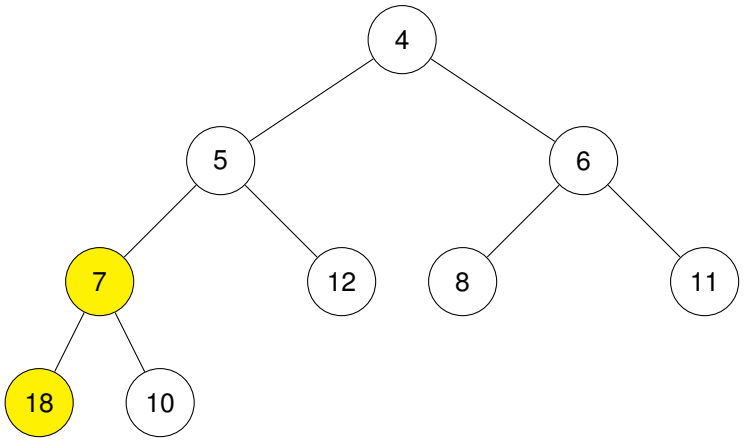
siftDown(18) :



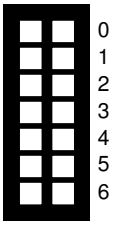
siftDown(18) :



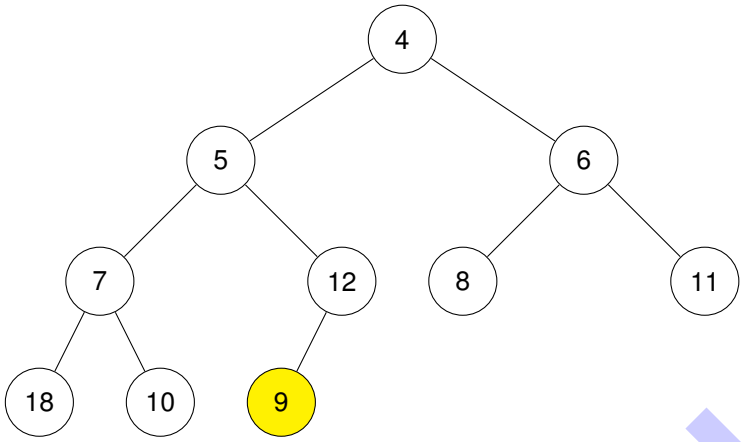
siftDown(18) :



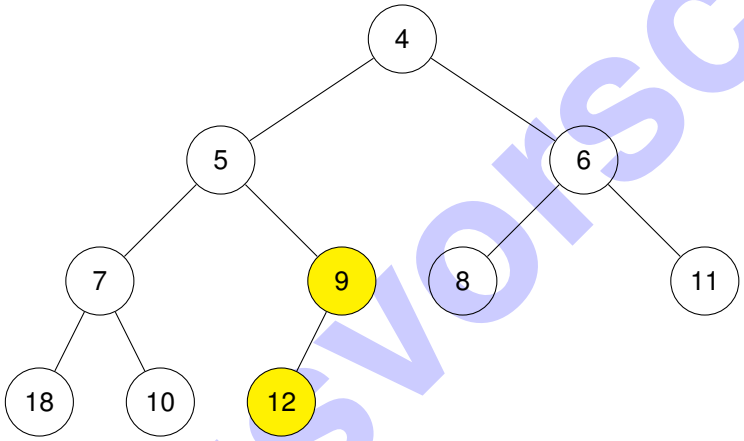
d) Fügen Sie nun auf dem neuen Heap die Zahl 9 ein. Geben Sie dabei alle nötigen Zwischenschritte an.



Element 9 ans Ende anhängen :



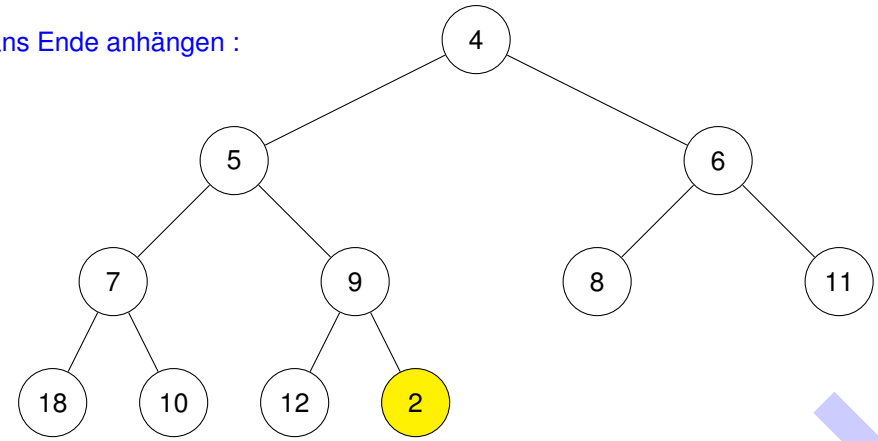
siftUp(9) :



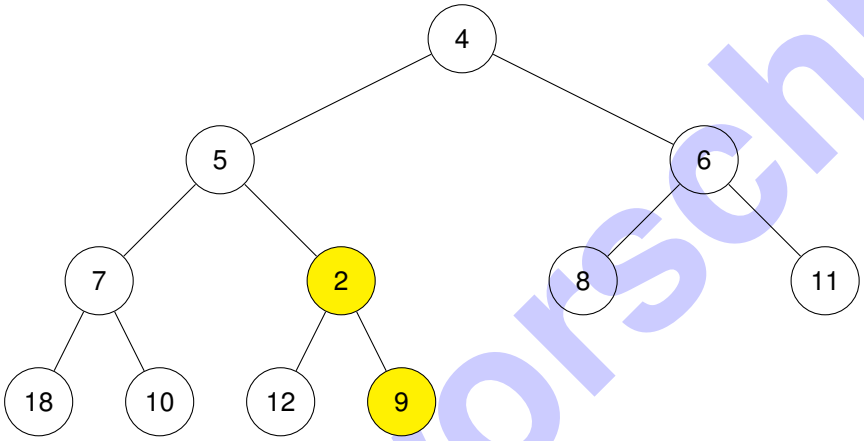
0
1
2
3
4
5
6

e) Fügen Sie nun auf dem neuen Heap die Zahl 2 ein. Geben Sie dabei alle nötigen Zwischenschritte an.

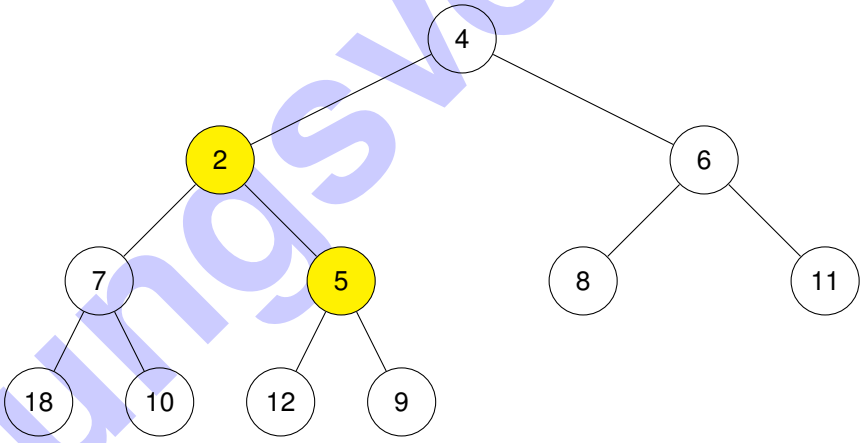
Element 2 ans Ende anhängen :



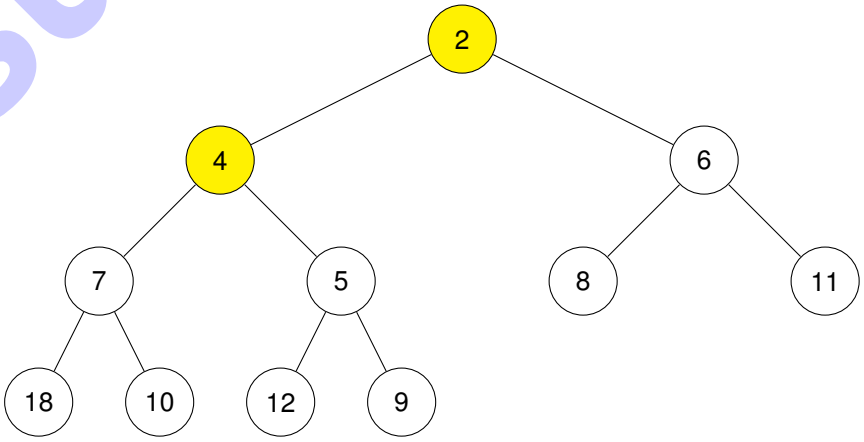
siftUp(2) :



siftUp(2) :



siftUp(2) :



f) Geben Sie abschließend wieder das Array an, das dem Heap nach allen von Ihnen ausgeführten Operationen entspricht.

0
1

2	4	6	7	5	8	11	18	10	12	9

Aufgabe 5 Dirty Double Hashing (20 Punkte)

Die Größe der Hashtabelle ist $m = 11$. Die Schlüssel der Elemente sind die Elemente selbst. Beim Löschen eines Elements soll das Element nur durch einen ”gelöscht”-Platzhalter (hier ’X’) ersetzt werden. Verwenden Sie die folgenden Hashfunktionen:

$$h(x, i) = (h(x) + i * h'(x)) \mod 11$$
$$h(x) = (x + 5) \mod 11$$
$$h'(x) = 1 + (q(x) \mod 10)$$

Hierbei bezeichnet $q(x)$ die Quersumme von x, also z.B. $q(123) = 1 + 2 + 3 = 6$ Beachten Sie, dass das Hashing bei jedem neuen Element jeweils mit $i = 0$ neu beginnt.

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

a) Führen Sie folgende Operationen in der gegebenen Reihenfolge aus. Geben Sie ebenfalls die überprüften Positionen im Feld "Position(en)"über der jeweiligen Tabelle an.

Insert: 15, 4, 20, 42, 27
Delete: 4, 42, 27
Insert: 42, 4

Nebenrechnungen:

Zahl	h(x)	h'(x)	h(x, 0)	h(x, 1)	h(x, 2)	h(x, 3)	h(x, 4)
4	9	5	9	3	8	2	7
15	9	7	9	5	1	8	4
20	3	3	3	6	9	1	4
27	10	10	10	9	8	7	6
42	3	7	3	10	6	2	9

1. Operation: Insert (15) **Position(en):** 9

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

2. Operation: Insert (4) **Position(en):** 9, 3

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

3. Operation: Insert (20) **Position(en):** 3, 6

0	1	2	3	4	5	6	7	8	9	10
			4			20			15	

4. Operation: Insert (42) **Position(en):** 3, 10

0	1	2	3	4	5	6	7	8	9	10
			4			20			15	42

5. Operation: Insert (27) **Position(en):** 10, 9, 8

0	1	2	3	4	5	6	7	8	9	10
			4			20		27	15	42

6. Operation: Delete (4) **Position(en):** 9, 3

0	1	2	3	4	5	6	7	8	9	10
			X			20		27	15	42

7. Operation: Delete (42) **Position(en):** 3, 10

0	1	2	3	4	5	6	7	8	9	10
			X			20		27	15	X

8. Operation: Delete (27) **Position(en):** 10, 9, 8

0	1	2	3	4	5	6	7	8	9	10
			X			20		X	15	X

9. Operation: Insert (42) **Position(en):** 3

0	1	2	3	4	5	6	7	8	9	10
			42			20		X	15	X

10. Operation: Insert (4) **Position(en):** 9, 3, 8

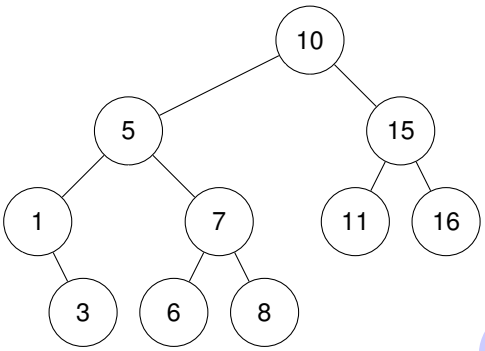
0	1	2	3	4	5	6	7	8	9	10
			42			20		4	15	X

Aufgabe 6 AVL-Bäume (20 Punkte)

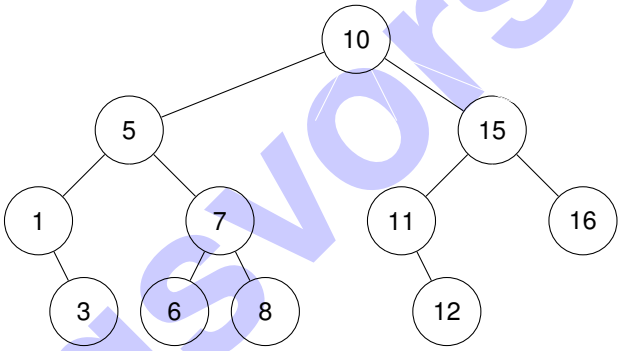
Führen Sie auf den folgenden AVL-Bäumen jeweils die angegebene Operation aus. Geben Sie im Lösungsfeld jeweils an, ob Sie keine, eine Einfach- oder eine Doppelrotation ausgeführt haben und deren jeweiligen Richtungen, und zeichnen Sie den fertigen Baum nach Ausführung der Operation hin. Sie können die Leerseiten am Ende des Klausurbogens nutzen, um nötige Zwischenschritte aufzuzeichnen. Denken Sie dabei bitte daran, jeweils die Aufgabennummer mit dazu anzugeben, damit die Zwischenschritte korrekt mit bewertet werden können.

0
1
2
3
4

a)* Fügen Sie den Knoten 12 ein.

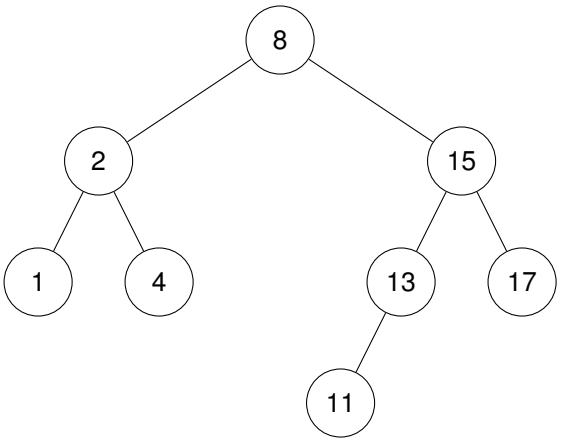


Keine Rotation

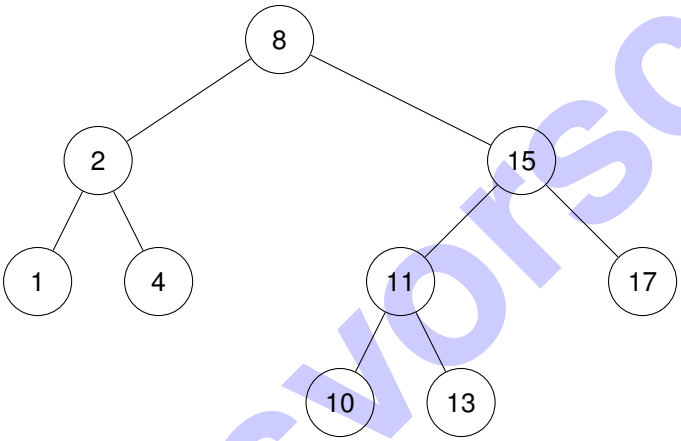


b)* Fügen Sie den Knoten 10 ein.

0 1 2 3 4

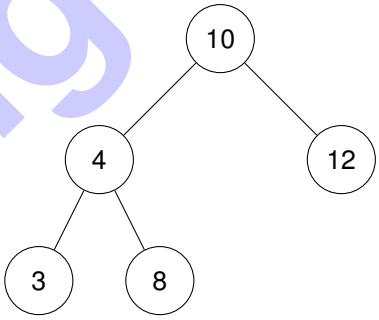


Einfachrotation rechts

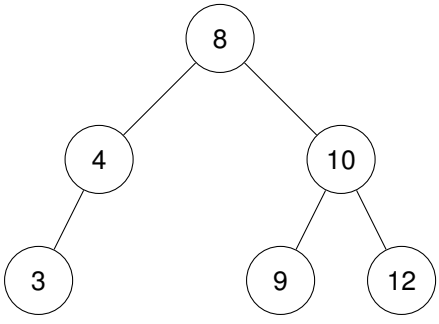


c)* Fügen Sie den Knoten 9 ein.

0 1 2 3 4

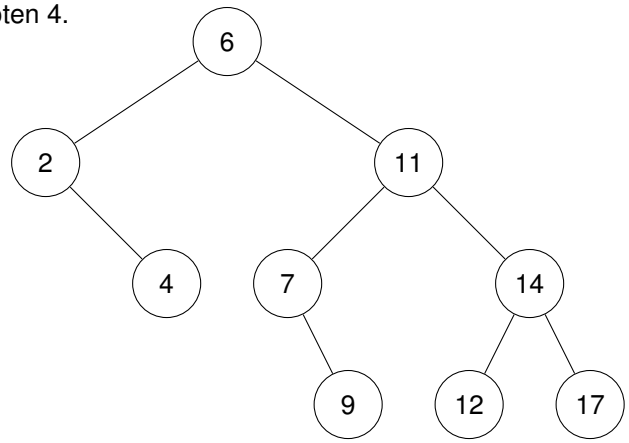


Doppelrotation links/rechts

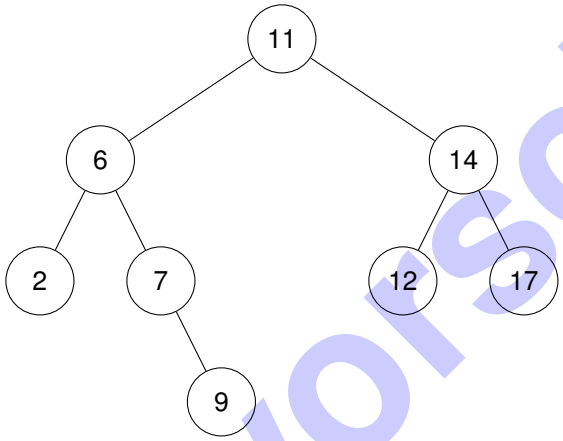


0
1
2
3
4

d)* Entfernen Sie den Knoten 4.

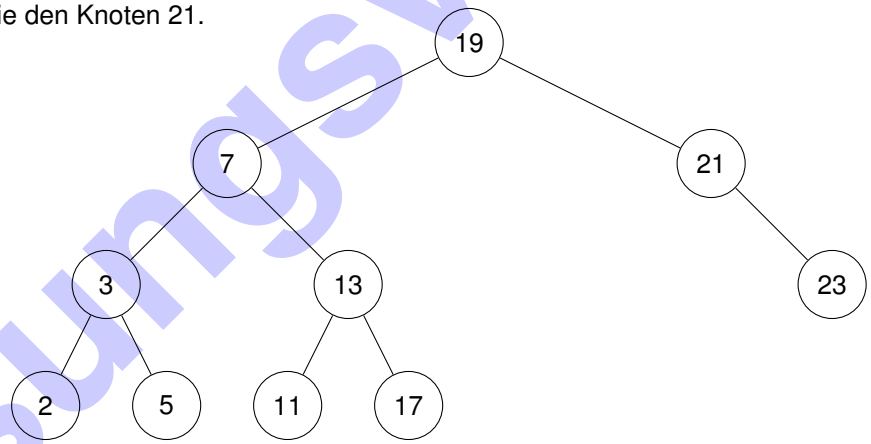


Einfachrotation links

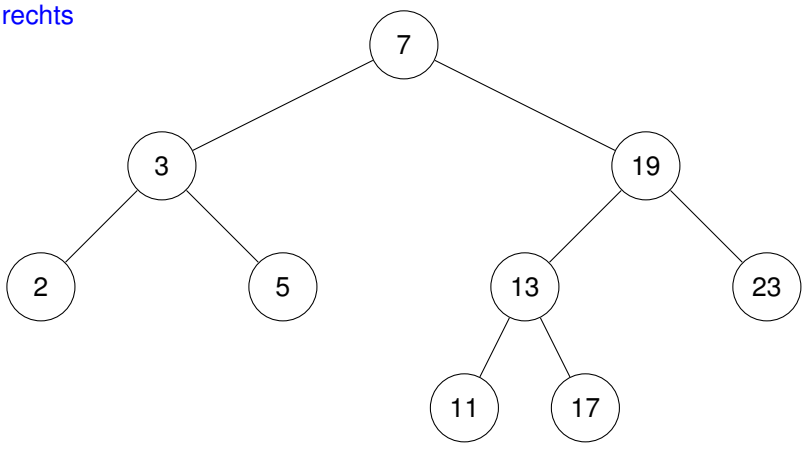


0
1
2
3
4

e)* Entfernen Sie den Knoten 21.

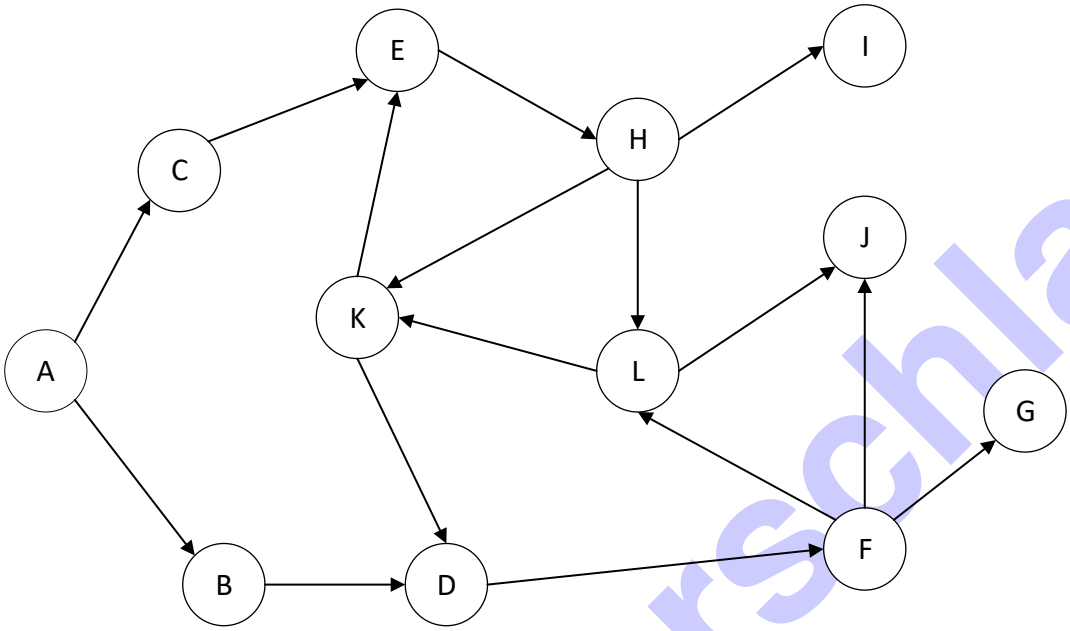


Einfachrotation rechts



Aufgabe 7 Tiefensuche in Graphen (20 Punkte)

Machen Sie sich mit folgendem Graphen vertraut:



a) Führen Sie eine Tiefensuche auf dem gegebenen Graphen durch. Starten Sie hierfür bei Knoten **A**. Gibt es bei einem Knoten mehrere Möglichkeiten für den Nachfolger, so wählen Sie bitte stets den Knoten als Nachfolger, der im Alphabet als nächstes folgt. Geben Sie in der nachfolgenden Tabelle für jeden Knoten, wie in der Vorlesung definiert, die `dfsNum` sowie die `finishNum` an.

0 1 2 3 4 5 6 7 8 9 10 11 12

Knoten	**A**	**B**	**C**	**D**	**E**	**F**
dfsNum	1	2	12	3	9	4
finishNum	12	10	11	9	5	8

Knoten	**G**	**H**	**I**	**J**	**K**	**L**
dfsNum	5	10	11	6	8	7
finishNum	1	4	3	2	6	7

Geben Sie für die nachfolgenden Kanten jeweils an, ob es sich um eine *Baum- oder Vorwärtskante*, eine *Rückwärtskante* oder *Kreuzkante* handelt.

b) Die Kante H $\rightarrow$ L ist eine:

- ☐ Baum- oder Vorwärtskante
- ☒ Rückwärtskante
- ☐ Kreuzkante

c) Die Kante L $\rightarrow$ K ist eine:

- ☒ Baum- oder Vorwärtskante
- ☐ Rückwärtskante
- ☐ Kreuzkante

d) Die Kante L $\rightarrow$ J ist eine:

- ☐ Baum- oder Vorwärtskante
- ☐ Rückwärtskante
- ☒ Kreuzkante

e) Die Kante E $\rightarrow$ H ist eine:

- ☒ Baum- oder Vorwärtskante
- ☐ Rückwärtskante
- ☐ Kreuzkante

f) Die Kante C $\rightarrow$ E ist eine:

- ☐ Baum- oder Vorwärtskante
- ☐ Rückwärtskante
- ☒ Kreuzkante

g) Die Kante K $\rightarrow$ D ist eine:

- ☐ Baum- oder Vorwärtskante
- ☒ Rückwärtskante
- ☐ Kreuzkante

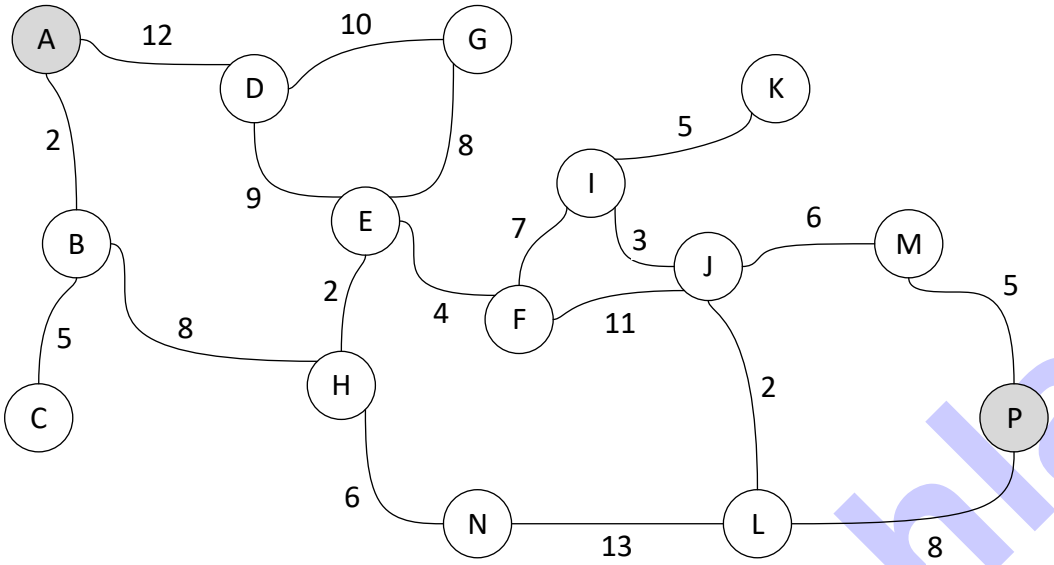
h) Die Kante F $\rightarrow$ L ist eine:

- ☒ Baum- oder Vorwärtskante
- ☐ Rückwärtskante
- ☐ Kreuzkante

i) Die Kante H $\rightarrow$ K ist eine:

- ☐ Baum- oder Vorwärtskante
- ☒ Rückwärtskante
- ☐ Kreuzkante

Aufgabe 8 Dijkstra (29 Punkte)



a) Gesucht ist der kürzeste Weg in obigem Graphen von Startknoten 'A' zum Zielknoten 'P'. Verwenden Sie hierzu den Algorithmus von Dijkstra, um die kürzesten Wege zu allen Knoten zu berechnen. Notieren Sie hierbei den Inhalt der Prioritätswarteschlange nach jedem Schritt des Algorithmus. Geben Sie außerdem jeweils an, welche Veränderungen Sie an der Prioritätswarteschlange ausführen. Bei Knoten mit gleicher Priorität bearbeiten Sie den Knoten zuerst, der zuerst in die Warteschlange eingefügt wurde.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Priority Queue	Updates der Priority Queue
(B, 2), (D, 12)	(B, 2), (D, 12)
(C, 7), (H, 10), (D, 12)	(C, 7), (H, 10)
(H, 10), (D, 12)	-
(D, 12), (E, 12), (N, 16)	(E, 12), (N, 16)
(E, 12), (N, 16), (G, 22)	(G, 22)
(N, 16), (F, 16), (G, 20)	(F, 16), (G, 20)
(F, 16), (G, 20), (L, 29)	(L, 29)
(G, 20), (I, 23), (J, 27), (L, 29)	(I, 23), (J, 27)
(I, 23), (J, 27), (L, 29)	-
(J, 26), (K, 28), (L, 29)	(J, 26), (K, 28)
(L, 28), (K, 28), (M, 32)	(L, 28), (M, 32)
(K, 28), (M, 32), (P, 36)	(P, 36)
(M, 32), (P, 36)	-
(P, 36)	-

b) Geben Sie hier schließlich den kürzesten Weg von 'A' nach 'P' und dessen Länge an.

A → B → H → E → F → I → J → L → P
Gesamtlänge 36

c) Der Algorithmus von Dijkstra lässt sich ggf. noch verbessern, wenn es möglich ist, in konstanter Zeit eine Heuristik zu berechnen, die angibt, wie weit es von einem bestimmten Knoten *mindestens* noch bis zum Zielknoten ist. Bei der Bestimmung der Priorität eines Knotens wird auf die bisher bestimmten Kosten der Wert der Heuristik aufaddiert. Diese (vereinfacht beschriebene) Variante ist auch unter dem Namen *A* Algorithmus* bekannt.
Verwenden Sie die folgende Tabelle, um Heuristikwerte von jedem Knoten zum Punkt 'P' zu bestimmen.

von	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P
nach P	28	27	31	26	19	16	19	20	9	8	9	6	4	16	0

Bestimmen Sie jetzt wie oben den kürzesten Pfad von 'A' nach 'P', allerdings unter Verwendung der hier beschriebenen Variante *A**. Bei Elementen in der Prioritätswarteschlange mit gleicher Priorität gehen Sie vor wie oben, bearbeiten also jeweils zuerst den ältesten Eintrag.

Priority Queue	Updates der Priority Queue
(B, 29), (D, 38)	(B, 2+27=29), (D, 12+26=38)
(H, 30), (D, 38), (C, 38)	(H, 30), (C, 38)
(E, 31), (N, 32), (D, 38), (C, 38)	(E, 31), (N, 32)
(N, 32), (F, 32), (D, 38), (C, 38), (G, 39)	(F, 32), (G, 39)
(F, 32), (L, 35), (D, 38), (C, 38), (G, 39)	(L, 35)
(I, 32), (L, 35), (J, 35), (D, 38), (C, 38), (G, 39)	(I, 32), (J, 35)
(J, 34), (L, 35), (K, 37), (D, 38), (C, 38), (G, 39)	(K, 37), (J, 34)
(L, 34), (M, 36), (K, 37), (D, 38), (C, 38), (G, 39)	(L, 34), (M, 36)
(M, 36), (P, 36), (K, 37), (D, 38), (C, 38), (G, 39)	(P, 36)
(P, 36), (K, 37), (D, 38), (C, 38), (G, 39)	-

Zusätzlicher Platz für Lösungen. Markieren Sie deutlich die Zuordnung zur jeweiligen Teilaufgabe. Vergessen Sie nicht, ungültige Lösungen zu streichen.