/ 20 P

Programmieraufgabe P1.

Passwort für Einschreibung: `AlgoDataExam2018`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Structure

In dieser Aufgabe wird ein Max-Heap implementiert. Ihre Aufgabe ist es, die drei folgenden Methoden zu vervollständigen.

- **`buildHeap()`**: baut einen Heap aus einem gegebenen Array von Schlüsseln in beliebiger Reihenfolge.
- **`insert(`**x**`)`**: fügt einen neuen Schlüssel x in den Heap.
- **`deleteMax()`**: löscht den grössten Schlüssel aus dem Heap.

Den grössten Teil der Implementierung des Max-Heaps ist bereits in der Programmvorlage vorhanden (inklusive Code um die Eingabe einzulesen, Platz für den Heap zu allozieren, den Zustand des Heaps auszugeben, etc).

Der Heap enthält N Schlüssel, die Ganzzahlen im Bereich $[-2^{31}, 2^{31} - 1]$ sind. Zur Vereinfachung nehmen wir an, dass der Heap nie mehr als 100'000 Schlüssel enthält, d.h. $N \leq 100000$.

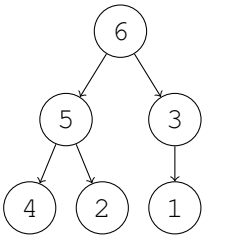
Um die Korrektheit Ihrer Implementierung zu überprüfen wird jeweils der Zustand des Heaps ausgegeben. Angenommen der Heap hält N Schlüssel, dann ist der Zustand des Heaps gegeben durch N partiell geordnete Ganzzahlen, welche die Heap Eigenschaft erfüllen. Der Einfachheit halber enthält das Template bereits eine Methode, die diesen Zustand ausgibt.

Beispiel

- Die **`buildHeap()`** Methode baut einen Heap durch das Wiederherstellen der Heapbedingung in einem gegebenen Array von Schlüsseln. Zum Beispiel für den Array aus $N = 5$ Schlüsseln

1	2	3	4	5	6

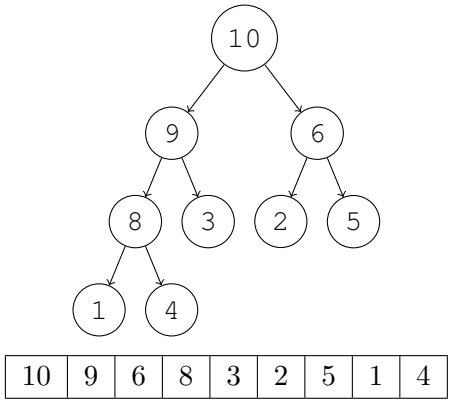
  ist der Zustand des Heaps sowie die partielle Ordnung der Zahlen wie folgt:

6	5	3	4	2	1

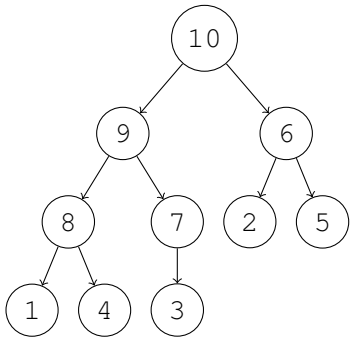
  Sobald der Heap gebaut ist, wird sein Zustand ausgegeben.

- Nehmen sie für die **`insert(`**x**`)`** Methode folgenden Heap aus $N = 9$ Schlüsseln an:



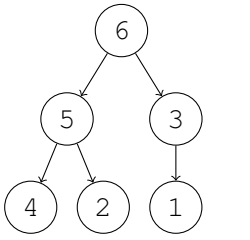
10	9	6	8	3	2	5	1	4

Das Einfügen des Schlüssels $x = 7$ in den Heap ergibt folgenden Heap aus $N = 10$ Schlüsseln:



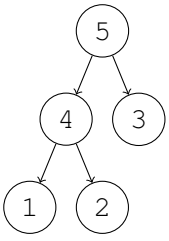
10	9	6	8	7	2	5	1	4	3

- Nehmen Sie für die **`deleteMax()`** Methode den folgenen Heap aus $N = 6$ Schlüsseln an:



6	5	3	4	2	1

Nach dem Löschen des grössten Schlüssels, sieht der Heap aus nun $N = 5$ Schlüsseln so aus:



5	4	3	1	2

Anforderungen

Insgesamt gibt es für diese Aufgabe maximal 20 Punkte auf dem Judge. Um die volle Punktzahl zu erhalten sollte ihr Programm $O(n)$ Zeit für die **`buildHeap()`** Methode und $O(\log(n))$ Zeit für die **`insert(`**x**`)`** und **`deleteMax()`** Methoden (alles mit sinnvollen versteckten Konstanten). Unvollständige Lösungen können Teilpunkte erzielen, nämlich:

- Sie können bis zu 8 Punkte erhalten für eine korrekte Implementierung von **`buildHeap()`**.
- Sie können bis zu 8 Punkte erhalten für eine korrekte Implementierung von **`insert(x)`**.
- Sie können bis zu 4 Punkte erhalten für eine korrekte Implementierung von **`deleteMax()`**.

Instruktionen

Wir stellen Ihnen für diese Aufgabe eine Programmvorlage als Eclipse Projekt zur Verfügung, das Sie beim Einlesen der Eingabe und dem Ausgeben der Ausgabe unterstützt. Das Importieren von zusätzlichen Java Klassen ist **nicht erlaubt** (mit Ausnahme der bereits importierten `java.io.{InputStream, OutputStream}` und `java.util.Scanner` Klassen).

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

Laden Sie ausschliesslich Ihr `Main.java` auf den Judge.

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

Eingabe Die Eingabe dieses Problems besteht aus mehreren Testinstanzen. Die erste Zeile enthält deren Anzahl T. Jede dieser T Testinstanzen ist unabhängig von den anderen, besteht aus einer Zeile und startet mit einer Befehlszahl, die entweder 1, 2 oder 3 ist.

1. Ist die Befehlszahl gleich 1, dann wird ein Array eingelesen, ein Heap daraus gebaut und der Zustand des Heaps ausgegeben.

   Die Befehlszahl ist gefolgt von einer Zahl N, die die Anzahl der Ganzzahlen im Array beschreibt und von diesen N Ganzzahlen im Bereich $[-2^{31}, 2^{31}-1]$. Alle Zahlen sind durch ein Leerzeichen getrennt.

2. Ist die Befehlszahl gleich 2, dann wird ein Heap eingelesen, danach werden Elemente in den Heap eingefügt und der Zustand des Heaps ausgegeben.

   Der Befehlszahl folgt die Grösse des Heaps N und N Ganzzahlen im Bereich $[-2^{31}, 2^{31}-1]$, welche die Heapbedingung erfüllen. Anschliessend folgt die Anzahl einzufügende Schlüssel M und M weitere Ganzzahlen im Bereich $[-2^{31}, 2^{31}-1]$. Alle Zahlen sind durch ein Leerzeichen getrennt.

3. Ist die Befehlszahl gleich 3, dann wird ein Heap eingelesen, danach wird mehrere Male der grösste Schlüssel aus dem Heap gelöscht und der finale Zustand des Heaps ausgegeben.

   Der Befehlszahl folgt die Grösse des Heaps N und N Ganzzahlen im Bereich $[-2^{31}, 2^{31} - 1]$, welche die Heapbedingung erfüllen. Anschliessend folgt eine Ganzzahl M, welche angibt wie oft das jeweils der grösste Schlüssel aus dem Heap gelöscht werden soll. Alle Zahlen sind durch ein Leerzeichen getrennt.

Ausgabe Bei allen Testinstanzen ist wird der finale Zustand des Heaps ausgegeben. Die Ausgabe enthält eine Zeile für jeden Test, wobei die i-te Zeile eine Folge von Zahlen getrennt durch Leerzeichen enthält, die dem Zustand des Heaps entsprechen. Die Ausgabe endet mit einem End-Line Zeichen.

Beispielseingabe:

```
3
1 6 1 2 3 4 5 6
2 9 10 9 6 8 3 2 5 1 4 1 7
3 6 6 5 3 4 2 1 1
```

Beispielsausgabe:

```
6 5 3 4 2 1
10 9 6 8 7 2 5 1 4 3
5 4 3 1 2
```

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

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: `AlgoDataExam2018`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Square

Gegeben sei eine 2-dimensionale binäre Matrix B mit M Zeilen und N Spalten ($0 \leq M, N \leq 1024$) gefüllt mit Nullen und Einsen. Ihre Aufgabe ist es die Fläche einer grössten quadratischen Teilmatrix, die ausschliesslich 1 enthält.

Beispiel

$$\begin{array}{ccccc} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{array}$$

$$\downarrow$$

$$\begin{array}{ccccc} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & \boxed{\mathbf{1}} & \boxed{\mathbf{1}} & 1 \\ 1 & 1 & \boxed{\mathbf{1}} & \boxed{\mathbf{1}} & 1 \\ 1 & 0 & 0 & 1 & 0 \end{array}$$

In der obigen $(M = 4) \times (N = 5)$ binären Matrix ist Fläche einer grössten quadratischen Teilmatrix, die nur 1 enthält, genau 4.

Anforderungen

Insgesamt gibt es für diese Aufgabe maximal 20 Punkte auf dem Judge. Um die volle Punktzahl zu erhalten sollte ihr Programm in $O(N \cdot M)$ Zeit laufen, mit sinnvollen versteckten Konstanten. Langsamere Lösungen können Teilpunkte erzielen, nämlich:

- Sie können bis zu 10 Punkte für eine $O(N^2 \cdot M^2)$-Zeit Lösung.
- Sie können bis zu 5 weitere Punkte erhalten (d.h. 15 Punkte insgesamt) für eine $O(N \cdot M \cdot \min(M, N))$-Zeit Lösung.
- Sie können bis zu 5 weitere Punkte erhalten (d.h. 15 Punkte insgesamt) für eine $O(N \cdot M)$-Zeit Lösung.

Instruktionen Wir stellen Ihnen für diese Aufgabe eine Programmvorlage als Eclipse Projekt zur Verfügung, das Sie beim Einlesen der Eingabe und dem Ausgeben der Ausgabe unterstützt. Das Importieren von zusätzlichen Java Klassen ist **nicht erlaubt** (mit Ausnahme der bereits importierten `java.io.{InputStream, OutputStream}` und `java.util.Scanner` Klassen).

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

Laden Sie ausschliesslich Ihr `Main.java` auf den Judge.

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

Eingabe Die Eingabe dieses Problems besteht aus mehreren Testinstanzen. Die erste Zeile enthält deren Anzahl T. Jede dieser T Testinstanzen ist unabhängig von den anderen und besteht aus mehreren Zeilen.

Die erste Zeile jedes Tests enthält die Ganzzahlen M und N getrennt durch ein Leerzeichen. Die nächsten M Zeilen enthalten N Ganzzahlen (jeweils entweder 0 oder 1) getrennt durch ein Leerzeichen.

Ausgabe Die Ausgabe jedes Tests enthält eine Ganzzahl auf einer separaten Zeile – die Fläche einer grössten quadratischen Teilmatrix aus Einsen.

Genauer gesagt enthält die i-te Zeile der Ausgabe eine einzelne Ganzzahl, die der Fläche einer grössten quadratischen Teilmatrix aus Einsen in der binären Martix des i-ten Tests entspricht.

Beispieleingabe:

```
1
4 5
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Beispielausgabe:

```
4
```

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

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

/ 14 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.

/ 2 P

a) Nachfolgend sehen Sie vier Folgen gewisser Momentaufnahmen von jeweils einem der fünf Algorithmen `InsertionSort` (Sortieren durch Einfügen), `SelectionSort` (Sortieren durch Auswahl), `QuickSort`, `MergeSort` und `BubbleSort`. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

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

_____Sort

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

_____Sort

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

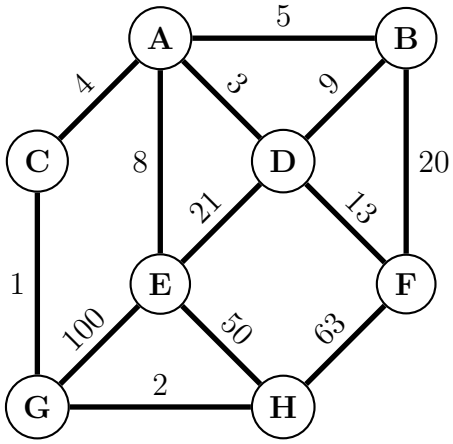
_____Sort

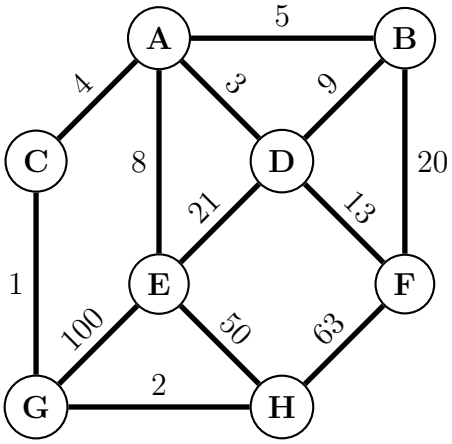
3	8	5	4	1	2	7	6
3	6	5	4	1	2	7	8
3	2	5	4	1	6	7	8

_____Sort

/ 2 P

b) Sie sehen unten zwei Kopien desselben Graphen. Markieren sie im linken Graphen die ersten 5 Kanten die Prims Algorithmus wählt, wenn er im Knoten D startet. Markieren Sie im rechten Graphen die ersten 5 Kanten, die Kruskals Algorithmus verwirft.



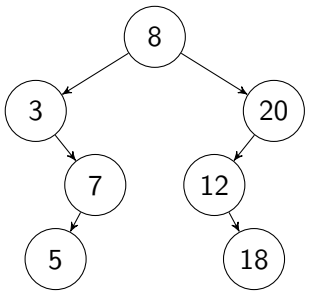


/ 1 P

c) *Binäre Suchbäume*: Zeichnen Sie den binären Suchbaum den Sie erhalten wenn Sie ausgehend von einem leeren Baum nacheinander die Schlüssel `3`, `8`, `12`, `5`, `20`, `7`, `18` einfügen.

/ 1 P

d) *Binäre Suchbäume*: Zeichnen Sie den binären Suchbaum den Sie erhalten wenn Sie aus folgendem binären Suchbaum den Schlüssel `8` löschen.

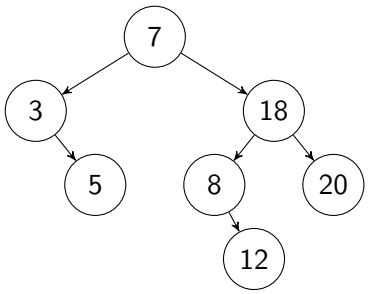


/ 1 P

e) *AVL-Bäume*: Zeichnen Sie den AVL-Baum den Sie erhalten wenn Sie ausgehend von einem leeren Baum nacheinander die Schlüssel `3`, `8`, `12`, `5`, `20`, `7`, `18` einfügen.

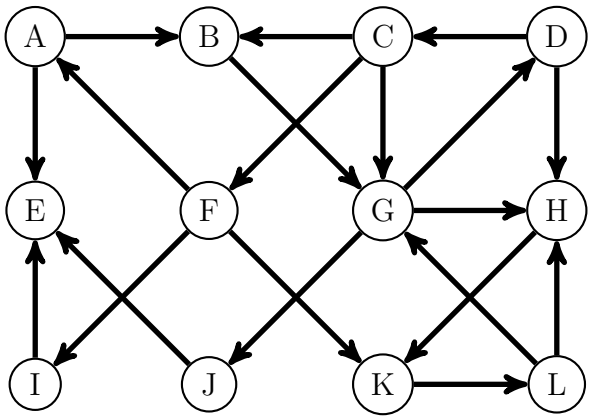
/ 1 P

f) *AVL-Bäume*: Zeichnen Sie den AVL-Baum den Sie erhalten wenn Sie aus folgendem AVL-Baum den Schlüssel `3` löschen.



/ 2 P

g) Führen Sie im folgenden Graphen je eine Tiefensuche und eine Breitensuche aus. Starten Sie in Knoten G und nehmen Sie an, dass die Traversierung die jeweiligen Nachbarknoten in alphabetischer Reihenfolge besucht. Geben Sie die Reihenfolgen (Tiefenordnung und Breitenordnung) an in welchen die Knoten besucht werden.



Tiefenordnung:

———, ———, ———, ———, ———, ———, ———, ———, ———, ———, ———, ———.

Breitenordnung:

———, ———, ———, ———, ———, ———, ———, ———, ———, ———, ———, ———.

/ 2 P

h) Geben Sie für die folgenden Behauptungen jeweils an, ob sie wahr oder falsch sind. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche werden 0,5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte.

Behauptung	**wahr**	**falsch**
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☐
$\log(n!) \geq \Omega(n^2)$	☐	☐
$n^k \geq \Omega(k^n)$, wenn $1 < k \leq O(1)$	☐	☐
$\log_3 n^4 = \Theta(\log_7 n^8)$	☐	☐

/ 2 P

i) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 9 \cdot T(\frac{n}{3}) + 32n - 16 & n > 1 \text{ und } n \text{ eine Dreierpotenz} \\ 6 & n = 1 \end{cases}$$

Sie können annehmen, dass n eine Potenz von 3 ist. Zeigen Sie mit vollständiger Induktion, dass

$$T(n) = 20n^2 - 16n + 2$$

die dazugehörige geschlossene Form ist.

Induktionsbeweis:

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

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

/ 14 P

Theorieaufgabe T2.

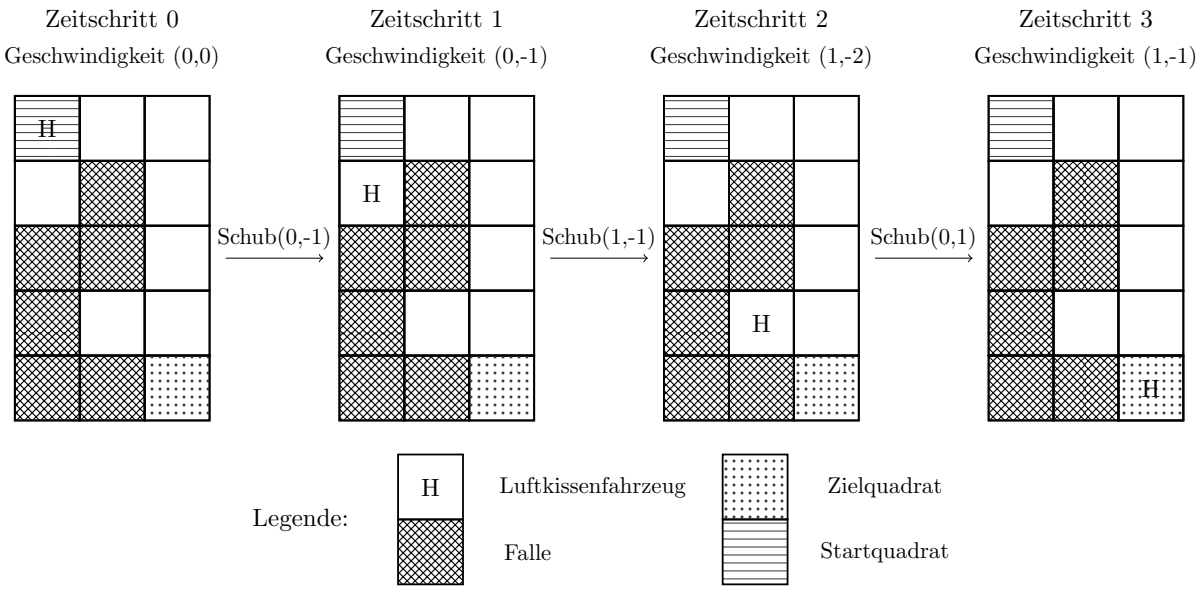
Sie fahren ein ferngesteuertes Luftkissenfahrzeug auf einem Feld. Das Feld ist ein Raster aus $m \times n$ Quadraten, und Sie möchten ein *Zielquadrat* von einem *Startquadrat* aus erreichen. Einige Quadrate sind Fallen und haben Stacheln, die aus dem Boden auftauchen und Ihr Luftkissenfahrzeug zerstören können. Die Positionen dieser Fallen sind bekannt. Das Luftkissenfahrzeug wird auch zerstört, wenn es das $m \times n$ Feld verlässt. Die Reibung zwischen Ihrem Luftkissenfahrzeug und dem Boden ist sehr gering, daher ist es schwierig, die Geschwindigkeit zu ändern. Um die Geschwindigkeit und Richtung des Luftkissenfahrzeug zu ändern, können Sie dessen Triebwerke (Schubdüsen) verwenden.

Das Luftkissenfahrzeug hat eine Position (x, y), $x \in \{1, \ldots, m\}$ und $y \in \{1, \ldots, n\}$, und eine Geschwindigkeit (v_x, v_y), $v_x, v_y \in \{-2, -1, 0, 1, 2\}$. Die Bewegung des Luftkissenfahrzeugs erfolgt nach Zeitschritten. Zu jedem Zeitschritt werden die folgenden Aktionen (in dieser Reihenfolge) ausgeführt:

1. Das Luftkissenfahrzeug kann eine Schubdüse entlang der x-Achse abfeuern. Dann wird $v_x := v_x + a_x$, wobei $a_x \in \{-1, 1\}$. (Dies ist nur zulässig, wenn die neue Geschwindigkeit gültig ist.)
2. Das Luftkissenfahrzeug kann eine Schubdüse entlang der y-Achse abfeuern. Dann wird $v_y := v_y + a_y$, wobei $a_y \in \{-1, 1\}$. (Dies ist nur zulässig, wenn die neue Geschwindigkeit gültig ist.)
3. Die Position des Luftkissenfahrzeugs wird aktualisiert: $x := x + v_x$ und $y := y + v_y$
4. Befindet sich das Luftkissenfahrzeug er einer Falle, dann treten die Stacheln hervor und das Luftkissenfahrzeug wird zerstört. Die Stacheln ziehen sich anschliessend wieder zurück.

Zu Beginn (Startquadrat) steht das Luftkissenfahrzeug still. Beim Erreichen seines Ziels darf es eine beliebige Geschwindigkeit haben. Ausserdem dürfen Sie annehmen, dass es immer möglich ist, das Zielquadrat zu erreichen.

Beispiel



/ 6 P

a) Ihr Ziel ist es, die Zahl der Zeitschritte zu minimieren, die erforderlich sind, um vom Startquadrat zum Zielquadrat zu gelangen. Modellieren Sie den Zustandsraum als Graphen $G = (V, E)$. Beschreiben Sie, was die Knoten und Kanten darstellen. Geben Sie für jeden Knoten $v \in V$ genau an, zu welchen anderen Knoten er eine Kanten hat. Nennen Sie einen möglichst effizienten Algorithmus zur Lösung dieses Problems, und geben Sie die Laufzeit des Algorithmus an in Form von m und n an.

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

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

Möglichst effizienter Algorithmus:

Laufzeit (in möglichst knapper Θ-Notation in n und m). Begründen Sie Ihre Antwort:

/ 4 P

b) Wir modifizieren das Problem jetzt so, dass die Schubdüsen des Luftkissenfahrzeugs von einer Patrone aus komprimiertem Kohlendioxid angetrieben werden. Diese Schubdüsen können nur T-mal verwendet werden, wobei T eine Konstante ist. Ihr Ziel ist es noch immer, die Zahl der Zeitschritte zu minimieren, die erforderlich sind, um vom Startquadrat zum Zielquadrat zu gelangen. Modellieren Sie für dieses modifizierte Problem den Zustandsraum als Graphen $G = (V, E)$. Beschreiben, was die Knoten und Kanten darstellen, und geben Sie für jeden Knoten $v \in V$ genau an, zu welchen anderen Knoten er eine Kanten hat. (Ein Schub in beide Richtungen x und y kostet genauso viel wie ein Schub in eine Richtung.) Benennen Sie anschließend einen möglichst effizienten Algorithmus zur Lösung dieses Problems und geben Sie die Laufzeit des Algorithmus in Form von m und n an.

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

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

Möglichst effizienter Algorithmus:

Laufzeit (in möglichst knapper Θ-Notation in n und m). Begründen Sie Ihre Antwort:

/ 4 P

c) Wir modifizieren wiederum das Ursprungsproblem. Anstatt die Zahl der Zeitschritte bis zum Erreichen des Ziels zu minimieren, geben einige Quadrate jetzt Punkte, wenn das Luftkissenfahrzeug darauf einen Zeitschritt beendet. Die meisten Quadrate nehmen allerdings Punkte weg. Das Ziel ist es, die maximale Anzahl von Punkten zu erzielen, während Sie vom Startquadrat zum Endquadrat fahren.

Nehmen Sie dafür Folgendes an:

- Sie dürfen nur auf dem Startquadrat und dem Zielquadrat anhalten (d.h. Geschwindigkeit $(0, 0)$ haben).
- Das Startquadrat gibt negative Punkte.
- Eine optimale Lösung existiert und hat eine endliche Länge.

Modellieren Sie für dieses modifizierte Problem den Zustandsraum als Graphen $G = (V, E)$. Beschreiben Sie, was die Knoten und Kanten darstellen, und geben Sie für jeden Knoten $v \in V$ genau an, zu welchen anderen Knoten er eine Kanten hat. Nennen Sie einen möglichst effizienten Algorithmus zum Lösen dieses modifizierten Problems und geben Sie die Laufzeit des Algorithmus in Form von m und n an.

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

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

Schema geht auf der nächsten Seite weiter.

Möglichst effizienter Algorithmus:

Laufzeit (in möglichst knapper Θ-Notation in n und m). Begründen Sie Ihre Antwort:

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

Theorieaufgabe T3. **/ 12 P**

Das Curriculum der Universität besteht aus n Kursen. Nehmen Sie zur Vereinfachung an, dass die Kurse durch ihre Nummern 1 bis n repräsentiert werden. Für jeden Kurs C gibt es eine Liste der vorausgesetzten Kurse, wobei diese Liste auch leer sein kann. Vor dem Besuch eines Kurses C müssen die Studenten mindestens einen der vorausgesetzten Kurse bestanden haben. Beachten Sie, dass eine Voraussetzung nicht transitiv ist: Wenn Kurs A eine Voraussetzung für Kurs B ist und Kurs B eine Voraussetzung für Kurs C ist, dann bedeutet dies nicht, dass A eine Voraussetzung für C ist. Sie können davon ausgehen, dass das Curricukum keine zirkulären Voraussetzungen enthält: Es gibt keine Folge von Kursen $(C_0, C_1, \ldots, C_{k-1}, C_k)$ so dass C_i eine Voraussetzung ist für C_{i+1} für jedes $i < k$ und ausserdem C_k eine Voraussetzung ist für C_0.

An dieser Universität sind die Kurse besonders schwierig, sodass Sie pro Semester nur einen Kurs bestehen können. Um das Material besser zu verstehen, möchten Sie ausserdem Kurse die einander voraussetzen direkt nacheinander besuchen. Wenn Sie im vorigen Semester einen Kurs C bestanden haben, dann können Sie im aktuellen Semester können nur Kurse besuchen, für die C eine Voraussetzung ist.

Wir nennen einen von Kurs T *erreichbar* von Kurs S, wenn es eine Folge von Kursen gibt

$$(S = C_0, C_1, \ldots, C_{k-1}, C_k = T),$$

so dass C_i eine Voraussetzung für C_{i+1} für jedes $i < k$ ist.

In dieser Aufgabe sind wir an folgenden Fragen interessiert:

a) Wie können wir die Kurse und Voraussetzungen als gerichteten Graphen modellieren?

b) Ist ein Kurs T von einem Kurs S aus erreichbar (für einen Spezialfall des Graphen)?

c) Wie viele verschiedene Folgen von Kursen gibt es um T zu besuchen, nachdem man S bestanden hat (für den allgemenenen Fall des Graphen)?

In den Aufgaben b) und c) können Sie annehmen, dass der gerichtete Graph so repräsentiert wird, dass Sie durch die direkten Nachfolger und Vorgänger eines Knotens u in $\mathcal{O}(\deg_+(u))$ und $\mathcal{O}(\deg_-(u))$ Zeit traversieren können, wobei $\deg_-(u)$ den Eingangsgrad und $\deg_+(u)$ den Ausgangsgrad des Knotens u bezeichnet.

Beispiel

Betrachten Sie das folgende Curriculum mit fünf Kursen: Linear Algebra, Multivariable Calculus, Linear Programming, Convex Optimization, Combinatorial Optimization. Linear Algebra hat keine vorausgesetzten Kurse, ist aber Voraussetzung für Multivariable Calculus und Linear Programming. Multivariable Calculus und Linear Programming ihrerseits sind Voraussetzungen für Convex Optimization und schliesslich ist Convex Optimization eine Voraussetzung für Combinatorial Optimization. (Linear Algebra ist hingegen keine Voraussetzung für Convex Optimization und Combinatorial Optimization.) Die folgende Tabelle zeigt eine Übersicht aller Voraussetzungen und Erreichbarkeiten:

Kurs	**Hat Voraussetzungen**	**Ist erreichbar von**
Linear Algebra	—	Linear Algebra
Multivariable Calculus	Linear Algebra	Multivariable Calculus Linear Algebra
Linear Programming	Linear Algebra	Linear Programming Linear Algebra
Convex Optimization	Multivariable Calculus Linear Programming	Convex Optimization Multivariable Calculus Linear Algebra Linear Programming
Combinatorial Optimization	Linear Programming	Combinatorial Optimization Linear Programming Linear Algebra

Beachten Sie dass die Anzahl verschiedener Kursfolgen von Linear Algebra zu Convex Optimization gleich zwei ist: Eine Folge ist *(Linear Algebra, Multivariable Calculus, Convex Optimization)*, die andere Folge ist *(Linear Algebra, Linear Programming, Convex Optimization)*.

/ 1 P

a) Modellieren Sie die n Kurse und ihre Voraussetzungen als gerichteten Graphen: Geben Sie eine präzise Beschreibung der Knoten und Kanten dieses Graphen $G = (V, E)$ (wenn möglich in Worten und nicht formal).

/ 4 P

b) Nehmen Sie an, dass es einen Einführungskurs I gibt, der keine Voraussetzungen hat und dass jeder Kurs von I aus erreichbar ist. Nehmen Sie ausserdem an, dass es für jeden Kurs genau eine Möglichkeit gibt, um ihn zu besuchen, nachdem man I bestanden hat, d.h. für jeden Kurs J gibt es genau eine Folge von Kursen $(I = C_0, C_1, \ldots, C_k = J)$, so dass C_i eine Voraussetzung ist für C_{i+1} für jedes $i < k$.

Geben Sie auf dem in a) definierten Graphen einen möglichst effizienten Algorithmus an, der den Graphen G als Eingabe nimmt und ihn so vorbereitet, dass eine Abfrage "ist D erreichbar von C?" in $\mathcal{O}(1)$ Zeit beantwortet werden kann (die Dauer der Abfrage ist unabhängig von der Grösse des Graphen). Geben Sie die Laufzeit Ihres Vorbereitungs-Algorithmus an.

Hinweis: *Denken Sie über die folgenden Fragen nach: Wie viele Knoten und Kanten gibt es im Graphen? Wie sieht der Graph aus?*

Vorbereitungs-Algorithmus:

$\mathcal{O}(1)$-Zeit Abfrage-Algorithmus:

Laufzeit des Vorbereitungs-Algorithmus (in möglichst knapper Θ-Notation in n). Begründen Sie Ihre Antwort:

/ 7 P

c) Nehmen Sie an, dass Sie die Anzahl Möglichkeiten $N(S,T)$ finden möchten, um den Kurs T zu besuchen, nachdem Sie Kurs S bestanden haben, wenn Sie Kurse nur nacheinander besuchen. Das bedeutet, $N(S,T)$ ist die Anzahl verschiedener Kursfolgen ($S = C_0, C_1, \ldots, C_{k-1}, C_k = T$), so dass C_i eine Voraussetzung für C_{i+1} ist für jedes $i < k$.

Geben Sie ein möglichst effizientes Dynamisches Programm an, das den Graphen G aus Aufgabe a) sowie die Kurse S und T als Input nimmt und die Lösung $N(S,T)$ berechnet. Behandeln Sie dabei folgende Aspekte und geben Sie die Laufzeit Ihres Dynamischen Programms an:

1) *Definition der DP-Tabelle:* Welche Dimensionen hat die Tabelle $DP[\ldots]$? Was ist die Bedeutung jedes Eintrags?

2) *Berechnung eines Eintrags:* Welche Werte der Tabelle werden wie initialisiert? Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?

3) *Berechnungsreihenfolge*: In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?

4) *Auslesen der Lösung:* Wie lässt sich die Lösung am Ende aus der Tabelle auslesen?

Hinweis: *Denken sie über folgende Frage nach: Welche Grapheigenschaft ist durch den Satz "Das Curriculum enthält keine zirkulären Voraussetzungen." impliziert?*

Grösse der DP Tabelle / Anzahl Tabelleneinträge: _____

Bedeutung eines Tabelleneintrages (in klar formulierten Worten):

$DP[\ldots\ldots\ldots\ldots]$: _____

_____

Berechnung eines Tabelleneintrags (Initialisierung und Rekursion):

Schema geht auf der nächsten Seite weiter.

Reihenfolge der Berechnung:

Berechnung von $N(S,T)$:

Laufzeit in präziser Θ-Notation in n und m, wobei m die Anzahl Kanten in G sei. Begründen Sie Ihre Antwort:

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

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

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

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

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