be-OI 2026 **Finale - JUNIOR** Samstag 14. März 2026	**Füllt diesen Rahmen bitte in GROSSBUCHSTABEN und LESERLICH aus** VORNAME : ..... NAME : ..... SCHULE :	O **Reserviert**

Finale der Belgischen Informatik-Olympiade 2026 (Dauer: **2 Stunden**)

Anweisungen, die du vor der Prüfung sorgfältig lesen musst.

1. **Warte auf das Startsignal**, bevor du die Blätter aus der Plastikhülle nimmst.
2. Du musst das Finale in derselben Sprache ablegen, in der du die Qualifikation abgelegt hast. **Überprüfe, ob diese Seite in der richtigen Sprache ist.** Falls nicht, melde es oder wechsle den Platz.
3. Überprüfe, ob du die **richtige Fragenreihe** erhalten hast, die oben in der Kopfzeile angegeben ist.
   - Für Schüler bis zum zweiten Jahr der Sekundarschule: Kategorie **Kadett**, **rosa** Blätter.
   - Für Schüler im dritten oder vierten Jahr der Sekundarschule: Kategorie **Junior**, **grüne** Blätter.
   - Für Schüler ab dem fünften Jahr der Sekundarschule: Kategorie **Senior**, **gelbe** Blätter.
4. Wenn das Startsignal gegeben wird, nimm die Blätter aus der Plastikhülle, aber **entferne die Heftklammern nicht.**
5. Gib deinen Namen, Vornamen und deine Schule **sehr gut lesbar in GROSSBUCHSTABEN und nur auf dieser Seite** an.
6. Trage deine **Antworten auf den farbigen Blättern** ein, die mit dieser Seite zusammengeheftet sind. Schreibe **gut lesbar** mit einem **Kugelschreiber oder Stift** in Blau oder Schwarz.
7. Verwende die weißen Blätter als Schmierblätter. **Vergiss nicht, deine Lösungen auf die farbigen Blätter zu übertragen.**
8. Die Prüfung dauert 2 Stunden und du musst bis zum Ende an deinem Platz bleiben. Wenn du früher fertig bist, rufe eine Aufsichtsperson, um deine Antworten abzugeben.
9. Wenn das Schlusssignal gegeben wird, musst du sofort deine Lösungen abgeben.
   - Lege die Blätter nicht zurück in die Plastikhülle.
   - **Lege die farbigen Blätter, immer noch zusammengeheftet**, in eine der dafür vorgesehenen Boxen.
   - **Gib die Plastikhülle zurück**, indem du sie in eine dafür vorgesehene Box legst.
   - Behalte die weißen Blätter.
10. Du darfst nur Schreibmaterial dabei haben. Taschenrechner, Smartphones, ... sind **verboten**.
11. Alle Codeauszüge in den Aufgaben sind in **Pseudo-Code**. Auf den folgenden Seiten findest du eine **Beschreibung** des Pseudo-Codes, den wir verwenden. Wenn du mit Code antworten musst, kannst du den **Pseudo-Code** oder eine **gängige Programmiersprache** (Java, C, C++, Pascal, Python, ...) verwenden. Syntaxfehler werden bei der Auswertung nicht berücksichtigt.

Die Belgische Informatik-Olympiade ist möglich dank der Unterstützung unserer Mitglieder:

Pseudo-Code Kurzreferenz

Daten werden in Variablen gespeichert, jede wird durch einen Namen identifiziert.
Der Name ermöglicht den Zugriff auf die Variable, um Daten zu speichern oder abzurufen.
Man verwendet ← als Zuweisungsoperator, um einen Wert in einer Variable zu speichern.
Beispiel: `n←100` speichert die ganze Zahl `100` in der Variable namens `n`.

Variablen und einfache Daten	Beispiele
ganze Zahl	`n←100` `m←-1`
reelle Zahl	`pi←3.1415` `z←0.0`
logischer Wert (Boolean)	`t←`**`true`** (wahr) `t←`**`false`** (falsch)

Man kann arithmetische Operationen mit Zahlen und Variablen durchführen.

Addition und Subtraktion mit + und − (Beispiel: `a+3`)
Multiplikation mit ⋆ oder × (Beispiel: `2⋆a⋆b`)
ganzzahlige Division: Quotient mit `/` oder `//` und Rest mit `%` (Beispiel: `14//3` ist gleich `4` und `14%3` ist gleich `2`)
nicht ganzzahlige Division `/`: selten verwendet, wird gegebenenfalls angegeben (Beispiel: `3.6/4.0` ist gleich `0.9`)
Potenz oder Exponent mit ^ (Beispiel: `x^3` ist gleich `x⋆x⋆x`)

Man verwendet häufig Variablen, um ein Ergebnis zu berechnen
und es dann in einer Variable zu speichern.
Manchmal wird das Ergebnis in einer der bei der Berechnung verwendeten Variablen gespeichert.
Der Rahmen rechts enthält Code, der dies veranschaulicht.
Nach der Ausführung dieses Codes gilt `x=25`, `a=4` und `b=10`.

```
a ← 3
b ← 5
x ← a⋆b + 10
a ← a + 1
b ← b⋆2
```

```
if (a < b)
    { p ← p+5 }
else
    { p ← p-2 }
c ← c-1
```

Die Anweisung **`if`** ermöglicht es, Code nur auszuführen, wenn eine Bedingung wahr ist.
Man kann optional die Anweisung **`else`** hinzufügen, um einen anderen Code nur auszuführen, wenn die Bedingung falsch ist.
Im Beispiel links, wenn die Zahl in der Variable `a` kleiner ist als die in der Variable `b`, dann wird `5` zur Variable `p` addiert,
andernfalls wird `2` von der Variable `p` subtrahiert.
Anschließend wird in jedem Fall `1` von der Variable `c` abgezogen.

Die auszuführenden Anweisungen in den verschiedenen Fällen werden eindeutig gekennzeichnet,
entweder durch geschweifte Klammern { } oder durch Einrückung.
Meistens wird beides verwendet: geschweifte Klammern und Einrückung, wie oben gezeigt.

Hier ist die Liste der gebräuchlichsten Vergleichsoperatoren.

= oder == ist gleich	< kleiner als	<= oder ≤ kleiner oder gleich	> größer als	>= oder ≥ größer oder gleich	!= oder ≠ ist ungleich

Man kann mehrere Bedingungen testen, indem man sie mit den logischen Operatoren **and**, **or**, **not** kombiniert.

Die Bedingung `(p` **`and`** `q)` ist wahr, wenn beide Bedingungen `p` und `q` wahr sind.
Die Bedingung `(p` **`or`** `q)` ist wahr, wenn mindestens eine der beiden Bedingungen wahr ist.
Die Bedingung **`not`** `(p)` ist wahr, wenn `p` falsch ist, und falsch, wenn `p` wahr ist.
Man kann den logischen Wert einer Bedingung in einer booleschen Variable speichern, um ihn später zu verwenden. Beispiel: `f←((a==5)` **`and`** `(b>=0))` **`or`** `((a<0)` **`and`** `(b!=10))`
Die Bedingung eines **`if`** ist manchmal eine einfache boolesche Variable. Beispiel: **`if`** `(f) {a←10}` **`else`** `{b←10}`

Manchmal müssen mehrere Daten in einer einzigen strukturierten Variable gespeichert werden.

Variablen und strukturierte Daten	**Beispiele**
Array, Liste, Vektor	`seq←[7,11,0,-4,9]`
Array, Matrix	`M←[[0,1,2],[3,-1,3],[0,0,5]]`
Paar	`coord←(1,7)`
Text	`n←"John"`

Die einzelnen Elemente einer strukturierten Variable werden durch einen Index gekennzeichnet, der in eckigen Klammern nach dem Variablennamen geschrieben wird.
Das erste Element einer strukturierten Variable `G` hat den Index `0` und wird als `G[0]` bezeichnet.
Das zweite Element hat den Index `1` und das letzte hat den Index `n-1`, wenn es insgesamt `n` Elemente gibt.
Die Anzahl der Elemente einer strukturierten Variable wird durch die Funktion `len()` zurückgegeben.
Beispiel: wenn `G=[5,9,12]`, dann ist `len(G)=3`, `G[0]=5`, `G[1]=9` und `G[2]=12`.
Das Array hat die Größe `3`, aber der höchste Index ist `2`.

Um Code zu wiederholen, zum Beispiel um die Elemente eines Arrays zu durchlaufen, kann man eine **`for`**-Schleife verwenden.

```
sum ← 0
n ← len(T)
for (i ← 0 to n - 1 step 1)
{
    sum ← sum + T[i]
}
```

Die Schreibweise **`for`** `(i←a` **`to`** `b` **`step`** `k)`
stellt eine Schleife dar,

- in der `i` mit dem Wert `a` beginnt,
- die wiederholt wird, solange `i`$\leq$`b`,
- die `i` am Ende jedes Schrittes um `k` erhöht.

Das Beispiel links berechnet die Summe der Elemente des Arrays `T`. Die Elemente werden einzeln zur Variable `sum` hinzugefügt.

Die in einer Schleife auszuführenden Anweisungen werden eindeutig gekennzeichnet, entweder durch geschweifte Klammern { },
oder durch Einrückung. Meistens wird beides verwendet: geschweifte Klammern und Einrückung, wie oben gezeigt.

Man kann eine Schleife auch mit der Anweisung **`while`** schreiben, die Code wiederholt, solange ihre Bedingung wahr ist.

Im Beispiel rechts wird eine positive ganze Zahl `n` durch `2` geteilt,
dann durch `3`, dann durch `4` ...
bis sie nur noch aus einer einzigen Ziffer besteht
(das heißt, bis `n<10`).

```
d ← 2
while (n ≥ 10) {
    n ← n/d
    d ← d+1
}
```

Frage 1 – Der verlorene Palast Episode 1: Zählen.

Archäologen haben einen alten Palast mit vielen Räumen entdeckt.
Aber es gibt Fallen, die sie nicht vermeiden können.
Sie haben Professor Zarbi zu Hilfe gerufen.

Der Professor hat schnell erkannt, dass man auf bestimmte gravierte Fliesen treten muss, um die Fallen zu deaktivieren.
In den Räumen sind die gravierten Fliesen immer in mehreren horizontalen Reihen angeordnet.
Die erste Reihe besteht aus einer einzigen gravierten Fliese, die *Spitze* genannt wird.

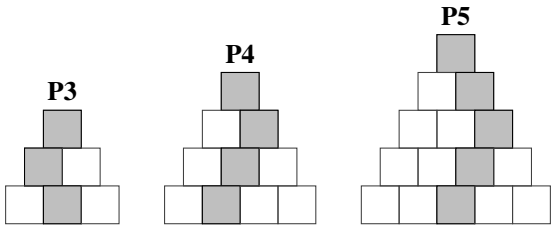
Der Professor hat entdeckt, dass man unter Einhaltung bestimmter Regeln vorangehen muss, sonst wird unweigerlich eine Falle ausgelöst.

1. Man muss an der Spitze starten (oben in den Beispielen) und nur auf gravierten Fliesen gehen.
2. Beim Vorangehen muss man immer von einer gravierten Fliese zu einer anderen wechseln, die sie berührt (nicht nur über eine Ecke).
3. Man muss genau eine gravierte Fliese in jeder Reihe betreten.
4. Man muss die letzte Reihe der gravierten Fliesen erreichen (ganz unten in den Beispielen).

Ab jetzt bedeutet das Wort ***Weg*** eine Durchquerung, die diese 4 Regeln einhält.
Ab jetzt sind alle erwähnten oder gezeichneten ***Fliesen*** gravierte Fliesen.

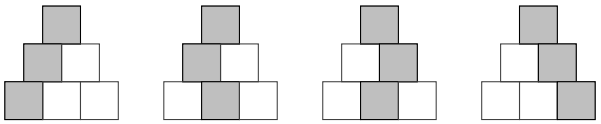
Die Fliesen sind oft als Pyramide angeordnet, wie in allen Beispielen auf dieser Seite.
Man bezeichnet mit **Pn** eine Pyramide, die aus n Reihen von Fliesen besteht.

Beispiel 1
Hier sind Pyramiden **P3**, **P4** und **P5**.
Ein Beispiel für einen Weg ist in jeder grau eingefärbt.
Die Spitze ist grau (Regel 1).
Die grauen Fliesen benachbarter Reihen berühren sich (Regel 2).
Es gibt genau eine graue Fliese pro Reihe (Regel 3).
Es gibt eine graue Fliese in der letzten Reihe (Regel 4).



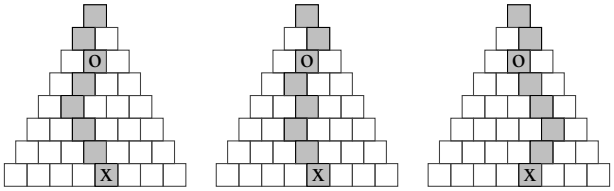
Professor Zarbi möchte zählen, wie viele verschiedene Wege es gibt, um Pyramiden zu durchqueren.

Beispiel 2
Hier sind alle Wege, die **P3** durchqueren.
1 Weg erreicht die linke Fliese der letzten Reihe.
2 Wege erreichen die mittlere Fliese der letzten Reihe.
1 Weg erreicht die rechte Fliese der letzten Reihe.
Insgesamt durchqueren 4 verschiedene Wege **P3**.



Manchmal möchte er nur die Wege zählen, die über bestimmte Fliesen führen.

Beispiel 3
Hier sind 3 Wege, die **P8** durchqueren.
Sie führen alle über die zweite Fliese der dritten Reihe
(Fliese mit einem o markiert).
Sie erreichen alle die fünfte Fliese der letzten Reihe
(Fliese mit einem x markiert).

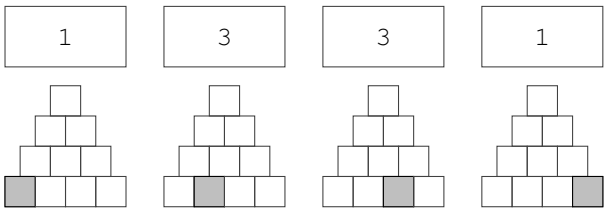


Hinweis: Auf einer der letzten Seiten dieses Fragebogens
stehen leere Pyramiden für deine Entwurfsarbeiten zur Verfügung.

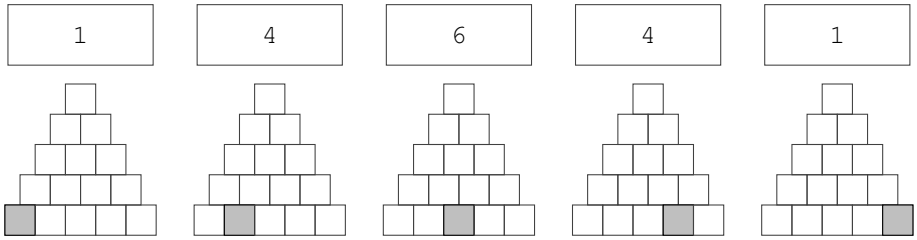
Wege zählen, die eine bestimmte Fliese erreichen.

In den folgenden Fragen sind Pyramiden mit 1 grauen Fliese in der letzten Reihe gezeichnet.
Wie viele verschiedene Wege erreichen die graue Fliese? Schreibe die Antworten in die Rechtecke.

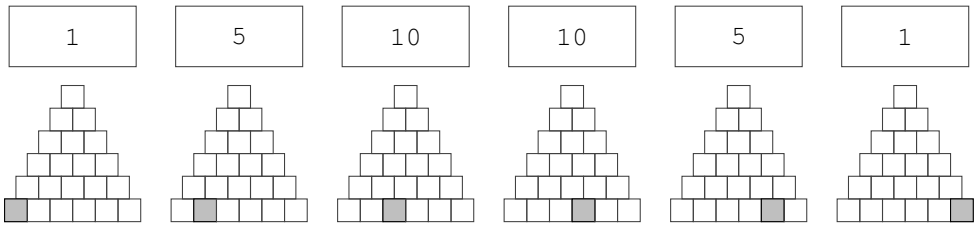
Q1(a) /4	**Wie viele Wege erreichen die graue Fliese in P4?**



Q1(b) /5	**Wie viele Wege erreichen die graue Fliese in P5?**

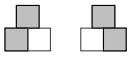


Q1(c) /6	**Wie viele Wege erreichen die graue Fliese in P6?**

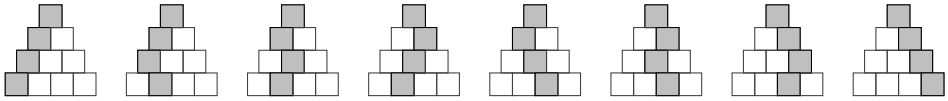


Alle Wege zählen.

Q1(d) /1	**Insgesamt, wie viele Wege durchqueren P2? Lösung : 2**



Q1(e) /1	**Insgesamt, wie viele Wege durchqueren P4? Lösung : 8**



Q1(f) /1	**Insgesamt, wie viele Wege durchqueren P5? Lösung : 16**

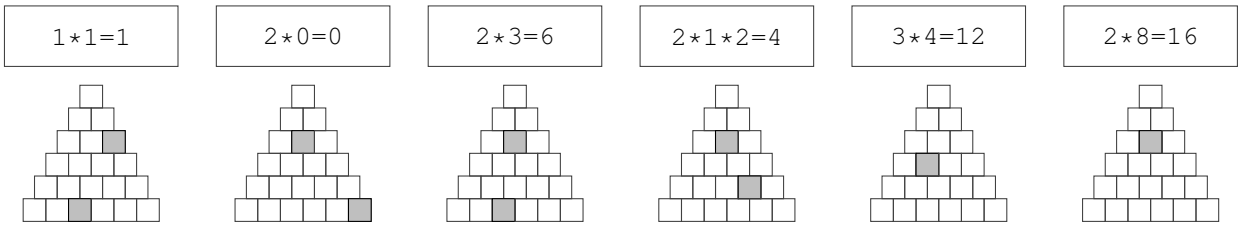
Q1(g) /1	**Insgesamt, wie viele Wege durchqueren P6? Lösung : 32**

Q1(h) /1	**Insgesamt, wie viele Wege durchqueren Pn (wobei n eine positive ganze Zahl ist)? Lösung :** 2^{n-1}

Wege zählen, die über bestimmte Fliesen führen.

In den folgenden Fragen sind Pyramiden mit einer oder mehreren grauen Fliesen gezeichnet.
Schreibe in das Rechteck über der Pyramide die Anzahl der verschiedenen Wege, die über die grauen Fliesen führen. **Vergiss die 4 Regeln nicht!**

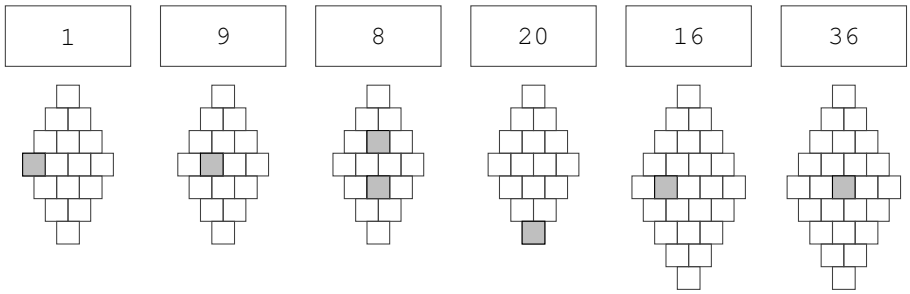
Q1(i) /6	Wie viele Wege führen über die grauen Fliesen der Pyramiden P6?



Fliesen in Diamantform angeordnet.

Die Fliesen sind oft als Pyramide angeordnet, aber das ist nicht immer der Fall.
Es gibt zum Beispiel eine Anordnung in Diamantform, bei der die letzte Reihe nur eine einzige Fliese hat.
In den folgenden Fragen sind Räume in Diamantform mit einer oder mehreren grauen Fliesen gezeichnet.
Schreibe in das Rechteck über dem Diamant die Anzahl der verschiedenen Wege, die über die grauen Fliesen führen.

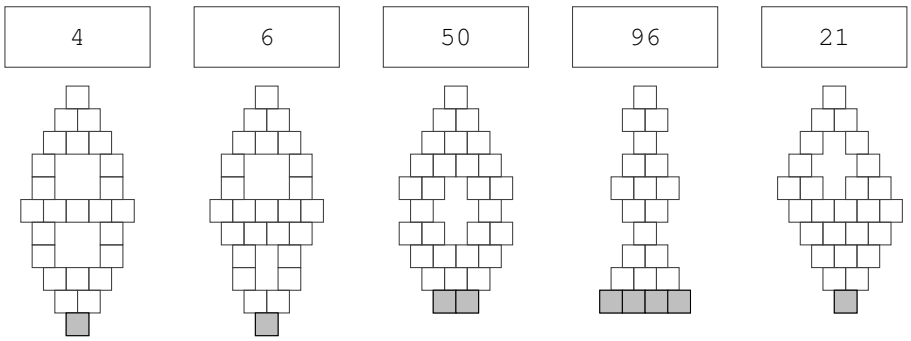
Q1(j) /6	Wie viele Wege führen über die grauen Fliesen der Diamanten?



Seltsame Räume.

Die Fliesen einiger Räume sind auf komplizierte Weise angeordnet.
In den folgenden Fragen müssen alle Wege gezählt werden, die seltsame Räume durchqueren.
Schreibe in das Rechteck über dem Raum die Gesamtzahl der Wege, die ihn durchqueren (bis zu einer grauen Fliese).

Q1(k) /5	Insgesamt, wie viele Wege durchqueren diese seltsamen Räume?



Schritte in großen Pyramiden.

Einige Räume enthalten sehr große Pyramiden aus Fliesen und Professor Zarbi möchte einen Weg finden, das Zählen der Wege zu vereinfachen.
Er verwendet die folgenden Konventionen.

- Die `n` Reihen von Fliesen einer Pyramide **Pn** sind von `0` bis `n-1` von oben nach unten nummeriert.
- In einer Pyramide hat die Reihe Nummer `r` immer `r+1` Fliesen, die von `0` bis `r` von links nach rechts nummeriert sind.
- Die Koordinaten einer Fliese sind `(r,c)`, wobei `r` ihre Reihe und `c` ihre Nummer in der Reihe ist.

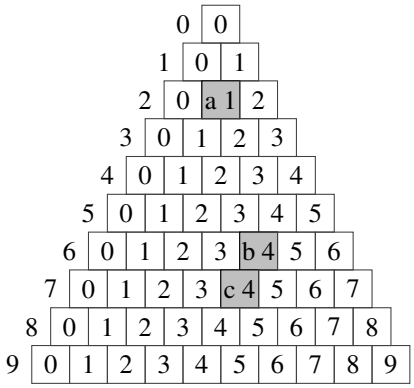
Beispiel

Die nebenstehende Abbildung zeigt eine Pyramide **P10**.
Die Reihennummern sind links angezeigt.
Die in jeder Reihe verwendeten Nummern sind auf den Fliesen angezeigt.
Die 3 grauen Fliesen sind mit den Buchstaben a, b und c gekennzeichnet.
Ihre Koordinaten sind `(2,1)`, `(6,4)` und `(7,4)`.

Hinweis

Diese Abbildung zeigt auch die Koordinaten
der Fliesen in kleineren Pyramiden.
Zum Beispiel zeigen die ersten 4 Reihen
die Koordinaten in einer Pyramide **P4**.

P10



Professor Zarbi zählt in mehreren Schritten die Wege, die über graue Fliesen einer großen Pyramide führen.
Jeder Schritt verwendet eine Teilpyramide.

- Die erste Teilpyramide hat dieselbe Spitze wie die große Pyramide,
  und ihre letzte Reihe enthält die erste graue Fliese der großen Pyramide.
- Danach verwendet man Teilpyramiden, deren Spitze eine graue Fliese ist,
  und deren letzte Reihe die nächste graue Fliese enthält.
- Es gibt einen Sonderfall, wenn die letzte Reihe der großen Pyramide keine graue Fliese enthält.
  In diesem Fall gibt es keine graue Fliese in der letzten Reihe der letzten Teilpyramide.

Professor Zarbi verwendet die folgenden Notationen.

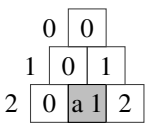
- `C(r,c)` ist die Anzahl der Wege, die die Fliese `(r,c)` erreichen (in einer Pyramide mit `r+1` Reihen).
- `T(n)` ist die Gesamtzahl der Wege, die eine Pyramide **Pn** durchqueren (also mit `n` Reihen).
  Achtung: Für die Zwecke dieses **BeOI**-Finales ist es **verboten, `T(1)` zu verwenden**.

Beispiel

Hier ist die Zerlegung in 4 Schritte der Wegzählung in **P10** oben.
Erster Schritt: von der Spitze zur ersten grauen Fliese a.
Zweiter Schritt: von der ersten grauen Fliese a zur zweiten b.
Dritter Schritt: von der zweiten grauen Fliese b zur dritten c.
Letzter Schritt: von der letzten grauen Fliese c zur letzten Reihe von **P10**.
Die Anzahl der möglichen Wege für jeden Schritt steht im Rechteck über dem Schritt.

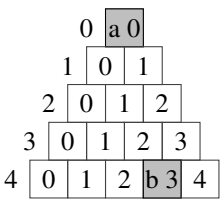
`C(2,1)`

P3



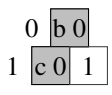
`C(4,3)`

P5



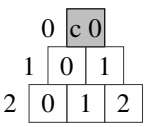
`C(1,0)`

P2



`T(3)`

P3

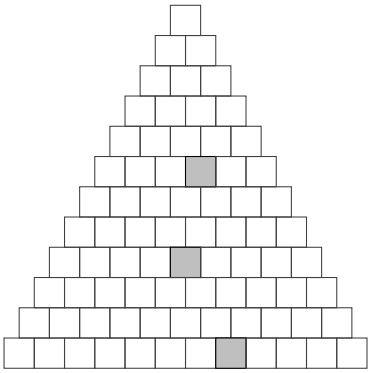


Natürlich muss man die Anzahlen der möglichen Wege der Schritte verwenden, um die Anzahl der Wege über die grauen Fliesen in der großen Pyramide zu bestimmen.
In den folgenden Fragen gib einen mathematischen Ausdruck mit den Notationen `C(r,c)` und `T(n)` an, um die Anzahl der Wege zu bestimmen, die über die grauen Fliesen des Beispiels **P10** von der vorherigen Seite und der Pyramiden **P12**, **P13** und **P15** unten führen.

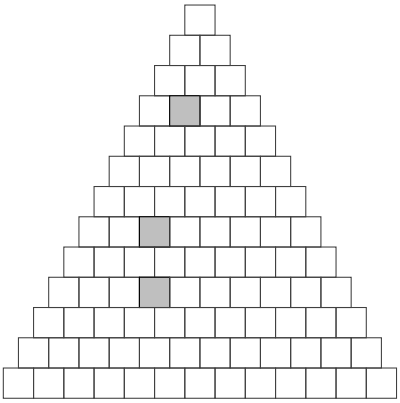
Wichtiger Hinweis
Berechne nicht die endgültige Antwort !
Du musst einen mathematischen Ausdruck schreiben, der zum Beispiel so aussieht: `C(5,2)*T(3)+C(4,2)`.

Q1(l) /2	**Ausdruck für die Anzahl der Wege über die grauen Fliesen von P10 aus dem Beispiel.** **Lösung :** `C(2,1) * C(4,3) * C(1,0) * T(3)`
Q1(m) /2	**Ausdruck für die Anzahl der Wege über die grauen Fliesen von P12.** **Lösung :** `C(5,3) * C(3,1) * C(3,3)`
Q1(n) /2	**Ausdruck für die Anzahl der Wege über die grauen Fliesen von P13.** **Lösung :** `C(3,1) * C(4,1) * C(2,1) * T(4)`
Q1(o) /2	**Ausdruck für die Anzahl der Wege über die grauen Fliesen von P15.** **Lösung :** `C(3,2) * C(4,2) * C(3,0) * C(3,1) * T(2)`

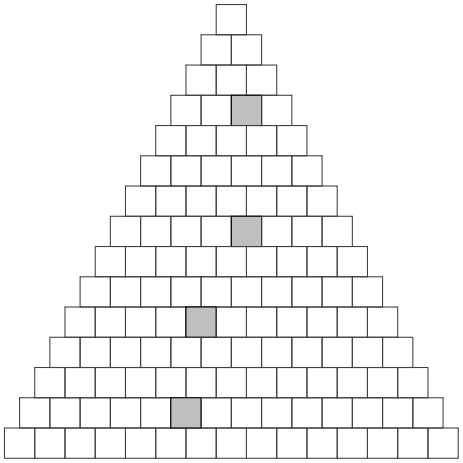
P12



P13



P15



Automatisierung.

Professor Zarbi hat eine Funktion `countPyra(n,G)` geschrieben, um das Zählen der Anzahl der Wege über graue Fliesen in einer großen Pyramide zu automatisieren.

Diese Funktion verwendet 2 korrekt initialisierte Parameter.

- `n`: eine ganze Zahl, die die Anzahl der Reihen der großen Pyramide enthält.
- `G`: ein Array von Paaren `(r,c)`, das die Koordinaten der grauen Fliesen enthält (in der Reihenfolge von oben nach unten).

Das Programm verwendet auch die Funktionen `C(r,c)` und `T(n)`, die die zuvor erklärten Werte zurückgeben.
Schließlich verwendet das Programm eine Schleife "**`for (r,c) in G`**" die so oft ausgeführt wird, wie es Elemente in `G` gibt.

Beispiele mit der Pyramide P10, die weiter oben in diesem Dokument gezeichnet ist.

- `n=10`
- `G=[(2,1), (6,4), (7,4)]`
  Also `G[0]=(2,1)`, `G[1]=(6,4)`, `G[2]=(7,4)`.
- `(r0,c0)←(0,0)` initialisiert die 2 Variablen gleichzeitig (das ist äquivalent zu `r0←0` und `c0←0`).
- `C(2,1)` gibt `2` zurück und `T(3)` gibt `4` zurück.
- Die Schleife "**`for (r,c) in G`**" wird 3-mal ausgeführt.
  Beim ersten Mal `(r,c)=(2,1)`, beim zweiten Mal `(r,c)=(6,4)`, beim letzten Mal `(r,c)=(7,4)`.

Professor Zarbi schreibt eine erste Version des Programms, die keine Überprüfung durchführt.
Dieses einfache Programm funktioniert korrekt mit den 4 großen Pyramiden **P10**, **P12**, **P13** und **P15**, vorausgesetzt, `n` und `G` werden korrekt initialisiert.

Das Programm verwendet die folgenden Variablen.

- `nbr`: die Anzahl der Wege über die grauen Fliesen der großen Pyramide.
- `h`: die Nummer der letzten Reihe der gerade bearbeiteten Teilpyramide (die also `h+1` Reihen hat).
- `k`: die Nummer der grauen Fliese in der letzten Reihe der aktuellen Teilpyramide (mit $0 \leq k \leq h$).

Die folgende Frage bringt zwischen 0 und 6 Punkte.
Du verlierst 2 Punkte für jedes Feld ____, das nicht korrekt ausgefüllt ist. Es gibt keine negativen Punkte.

Q1(p) /6	**Vervollständige die ____ in der Funktion `countPyraV1(n,G)`.**

```
function countPyraV1(n,G) {
  nbr ← 1
  (r0,c0) ← (0,0)

  for (r,c) in G {
    h ← r - r0

    k ← c - c0

    nbr ← nbr * C(h,k)

    (r0,c0) ← (r,c)
  }
  if (r0 < n-1) { nbr ← nbr * T(n-r0) }
  return nbr
}
```

Bestimmte Pyramiden können Probleme bereiten.

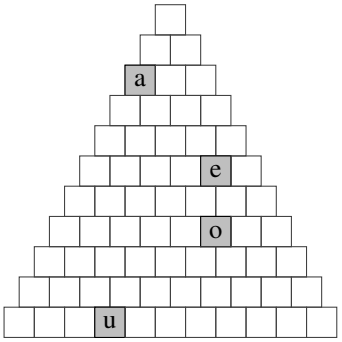
P11

Beispiel

Es gibt keine Wege, die über alle grauen Fliesen der nebenstehenden Pyramide führen.

Es ist unmöglich, von a nach e zu gelangen.

Ebenso gibt es keine Teilpyramide mit Spitze o, deren letzte Reihe u enthält.



Das Programm `countPyraV1` berücksichtigt diese Fälle nicht und kann falsche Antworten liefern oder mit einer Fehlermeldung abbrechen!
Das Programm `countPyraV2` ist eine Verbesserung, die in allen Fällen die richtige Antwort liefert.
Professor Zarbi hat lediglich die Codezeile `nbr←` aus `countPyraV1` durch einen Test in `countPyraV2` ersetzt.
Welche Bedingungen müssen überprüft werden und was muss geschehen, je nachdem ob sie wahr oder falsch sind?
Schreibe den Test in der folgenden Frage.

Diese Frage bringt zwischen 0 und 6 Punkte.
Du verlierst 2 Punkte für jedes Feld , das nicht korrekt ausgefüllt ist. Es gibt keine negativen Punkte.

Q1(q) /6	**Vervollständige die im Test von `countPyraV2`, der eine Zeile von `countPyraV1` ersetzt.**

```
if ( 0 <= k and k <= h )
    {nbr ← nbr * C(h,k) }
else
    {nbr ← 0 }
```

Frage 2 – Der verlorene Palast Episode 2: Entschlüsseln.

Die Entdeckungen von Professor Zarbi reichen noch nicht aus, um alle Fallen zu vermeiden.
Aber die Archäologen haben die Raumpläne zusammen mit anderen wichtigen Dokumenten gefunden.
In diesen Plänen ist auf jeder Fliese eine mit kleinen Scheiben markierte Pyramide gezeichnet.
Der Professor versteht, dass diese Pyramiden Zahlen darstellen, und es gelingt ihm, sie zu entschlüsseln.

Achtung! Nicht verwechseln!

In Episode 1 bestanden die Pyramiden aus Fliesen.

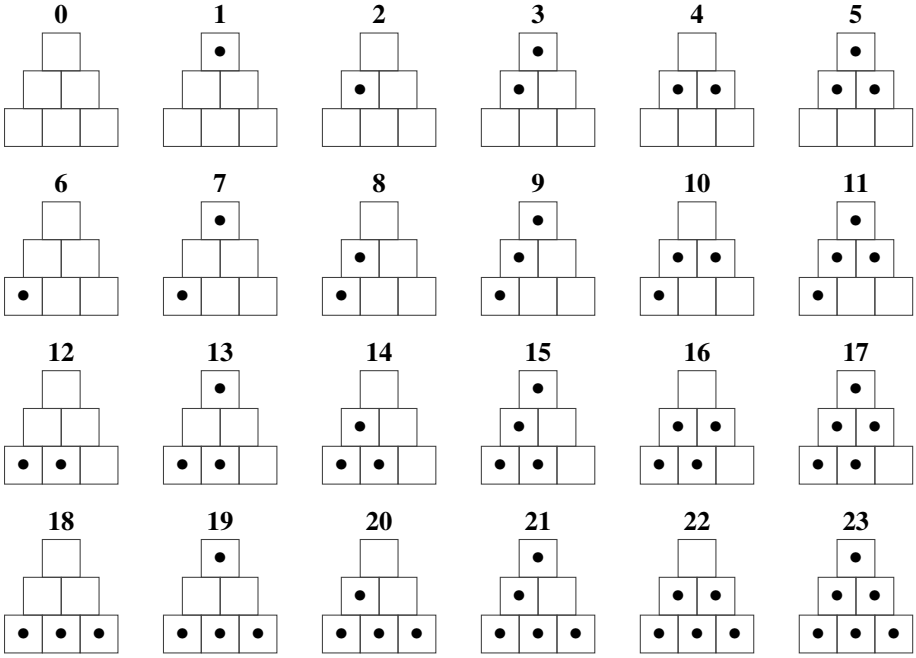
In Episode 2 sind die Pyramiden Zahlen, die auf den Fliesen markiert sind.

Das Nummerierungssystem.

Die Zahlen werden durch Pyramiden dargestellt, bei denen bestimmte Felder eine kleine schwarze Scheibe enthalten.

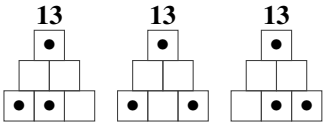
- 0 wird durch eine Pyramide ohne jede Scheibe dargestellt.
- Um von einer Pyramide zur nächsten zu gelangen, also von einer Zahl zur nächsten, geht man wie folgt vor.
  1. Von der Spitze ausgehend sucht man die erste Reihe, die ein leeres Feld enthält.
  2. Man setzt eine kleine Scheibe in ein beliebiges leeres Feld dieser Reihe.
  3. Man leert alle Felder der darüberliegenden Reihen (die gefüllt waren).

Beispiele: Hier sind Pyramiden **P3**, die die Zahlen von 0 bis 23 darstellen.



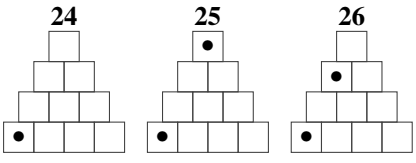
Zahlen mit mehreren Darstellungen.

Mehrere Pyramiden können dieselbe Zahl darstellen,
da man die schwarzen Scheiben beliebig
in ihrer Reihe platzieren kann (Punkt 2 oben).
Beispiel: Die 3 nebenstehenden Pyramiden stellen alle die Zahl 13 dar.



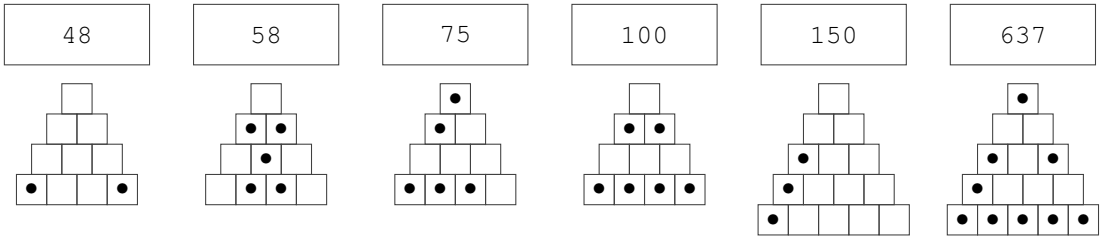
Darstellungen größerer Zahlen.

23 ist die größte Zahl, die man mit einer Pyramide **P3** darstellen kann.
Um weiterzukommen, muss man Pyramiden mit mehr Reihen verwenden.

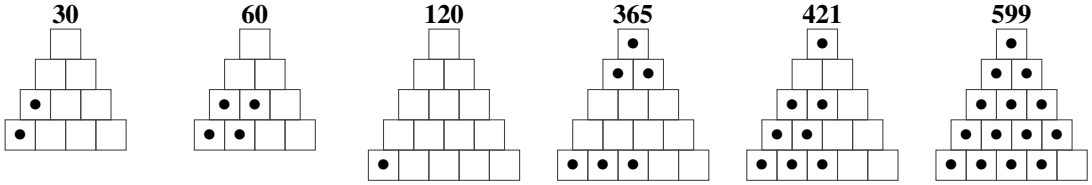


Hinweis: ***eine Pyramide zeichnen*** bedeutet, ***kleine schwarze Scheiben*** **in bestimmte Felder dieser Pyramide zu zeichnen**. Sonderfall: Wenn man keine Scheibe zeichnet, *zeichnet man eine leere Pyramide*, die die Zahl 0 darstellt.

Q2(a) /6	**Schreibe in jedes Rechteck die Zahl, die durch die gezeichnete Pyramide dargestellt wird.**



Q2(b) /6	**Zeichne eine Pyramide, die die angegebene Zahl darstellt.**



Oben wurde darauf hingewiesen, dass mehrere Pyramiden dieselbe Zahl darstellen können und dass 23 die größte Zahl ist, die man mit einer Pyramide **P3** darstellen kann. Daraus ergeben sich die folgenden Fragen, die du mit einer Zahl oder einem mathematischen Ausdruck beantworten kannst.

Q2(c) /1	**Wie viele verschiedene Pyramiden P3 kann man zeichnen?** Lösung : $2^6 = 64$

Q2(d) /1	**Wie viele verschiedene Pyramiden P4 kann man zeichnen?** Lösung : $2^{10} = 1024$

Q2(e) /1	**Was ist die größte Zahl, die man mit einer Pyramide P4 darstellen kann?** Lösung : $5! - 1 = 119$

Q2(f) /1	**Wie viele verschiedene Pyramiden P5 kann man zeichnen?** Lösung : $2^{15} = 32768$

Q2(g) /1	**Was ist die größte Zahl, die man mit einer Pyramide P5 darstellen kann?** Lösung : $6! - 1 = 719$

Q2(h) /1	**Wie viele verschiedene Pyramiden P6 kann man zeichnen?** Lösung : $2^{21} = 2097152$

Q2(i) /1	**Was ist die größte Zahl, die man mit einer Pyramide P6 darstellen kann?** Lösung : $7! - 1 = 5039$

In den folgenden Fragen musst du alle Zahlen angeben, die eine bestimmte Eigenschaft haben.
Wenn es mehrere Zahlen gibt, schreibe sie durch Kommas getrennt.
Wenn es keine solche Zahl gibt, antworte mit einem Kreuz $\times$.

Q2(j) /1	**Welche Zahlen haben genau eine Darstellung in P3?** Lösung : 0, 1, 4, 5, 18, 19, 22, 23

Q2(k) /1	**Welche Zahlen haben genau 2 Darstellungen in P3?** Lösung : 2, 3, 20, 21

Q2(l) /1	**Welche Zahlen haben genau 3 Darstellungen in P3?** Lösung : 6, 7, 10, 11, 12, 13, 16, 17

Q2(m) /1	**Welche Zahlen haben genau 4 Darstellungen in P3?** Lösung : X

Q2(n) /1	**Welche Zahlen haben genau 5 Darstellungen in P3?** Lösung : X

Q2(o) /1	**Welche Zahlen haben genau 6 Darstellungen in P3?** Lösung : 8, 9, 14, 15

Q2(p) /1	**Welche Zahlen haben genau 36 Darstellungen in P4?** Lösung : 56, 57, 62, 63

Q2(q) /1	**Was ist die kleinste Zahl, die genau 10 Darstellungen in P5 hat?** Lösung : 122

Q2(r) /1	**Was ist die größte Zahl, die genau 10 Darstellungen in P5 hat?** Lösung : 479

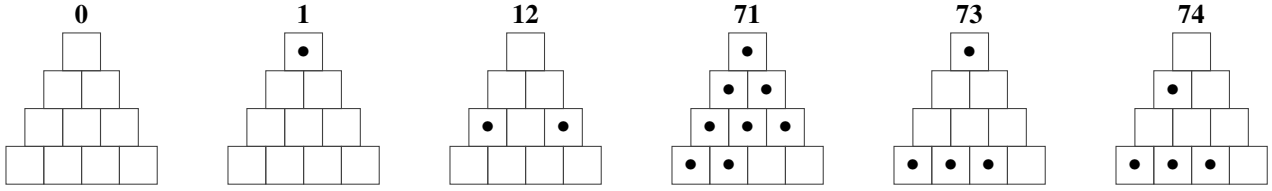
Neue Notation: Liste einer Pyramide.

Pyramiden zu zeichnen ist mühsam und unpraktisch.
Anstatt eine Pyramide zu zeichnen, gibt Professor Zarbi lieber die Liste der Anzahl schwarzer Scheiben in jeder Reihe an, beginnend von der Spitze.

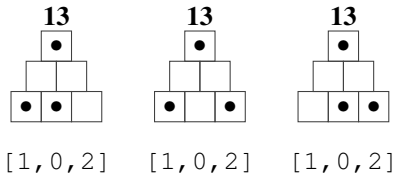
Ab jetzt ist die ***Zahl einer Pyramide*** die durch die Pyramide dargestellte Zahl.
Es handelt sich um eine positive ganze Zahl, wie üblich mit den Ziffern 0 bis 9 geschrieben.

Ab jetzt ist die ***Liste einer Pyramide*** die Liste der Anzahl schwarzer Scheiben in jeder Reihe der Pyramide.
Diese Liste beginnt mit `0`, wenn es keine schwarze Scheibe an der Spitze gibt, oder mit `1`, wenn es eine gibt.
Diese Liste enthält `n` Zahlen für eine Pyramide **Pn**.

Beispiele: Hier sind einige Pyramiden **P4**.
Für jede ist ihre **Zahl** oben und ihre **Liste** unten angegeben.



Bemerkung.
Wenn mehrere Pyramiden dieselbe Zahl darstellen,
dann haben sie dieselbe Liste.
Beispiel: Die 3 Pyramiden **P3**, die die Zahl 13 darstellen,
haben dieselbe Liste `[1,0,2]`.



Automatisierung.

Der Professor muss häufig die Zahl einer Pyramide aus ihrer Liste berechnen.
Umgekehrt muss er manchmal auch die Liste einer Pyramide aus ihrer Zahl berechnen.
Er möchte Programme schreiben, um seine Berechnungen zu automatisieren.

Einige Erinnerungen.

- **Ganzzahldivision.**
  Der Operator `/` (oder `//`, wenn du möchtest) gibt den ganzzahligen Quotienten einer Division zweier ganzer Zahlen.
  Der Operator `%` gibt den Rest einer Division zweier ganzer Zahlen.
  Beispiel: `14/3` ist gleich `4` (ebenso `14//3`) und `14%3` ist gleich `2`.
- **Nummerierung der Elemente einer Liste.**
  Die Elemente einer Liste sind ab `0` nummeriert.
  Beispiel: Wenn `LIS=[1,0,2]`, dann ist `LIS[0]=1`, `LIS[1]=0` und `LIS[2]=2`.
- **Initialisierung einer Liste.**
  `LIS←[0]*n` erstellt eine Liste mit `n` Nullen.

Funktion `PyrLIS(n,dec)`
Diese Funktion erhält 2 Parameter.

- `n`: die Anzahl der Reihen der Pyramide.
- `dec`: eine positive ganze Zahl, die durch eine Pyramide dargestellt werden soll.
  `dec` ist so benannt, um daran zu erinnern, dass es sich um eine **dezimale** Zahl handelt (d. h. mit den 10 Ziffern von 0 bis 9 geschrieben).

Die Funktion gibt die Liste einer Pyramide **Pn** zurück, die die Zahl `dec` darstellt.

Beispiele: `PyrLIS(3,13)` gibt `[1,0,2]` zurück und `PyrLIS(4,13)` gibt `[1,0,2,0]` zurück.

Die folgende Frage bringt zwischen 0 und 6 Punkte.
Du verlierst 3 Punkte für jede Zone , die nicht korrekt ausgefüllt ist.

Q2(s) /6	**Vervollständige die in der Funktion `PyrLIS(dec,n)`.**

```
function PyrLIS(n,dec) {
  LIS ← [0]*n
  for r ← 0 to n-1 step 1 {
    LIS[r] ← dec % (r+2)

    dec ← dec // (r+2)
  }
  return(LIS)
}
```

Funktion `Pyrdec(n,LIS)`
Diese Funktion erhält 2 Parameter.

- `n`: die Anzahl der Reihen der Pyramide.
- `LIS`: die Liste einer Pyramide.

Die Funktion gibt die Zahl `dec` der Pyramide **Pn** zurück, deren Liste gegeben ist.

Beispiele: `Pyrdec(3,[1,0,2])` und `Pyrdec(4,[1,0,2,0])` geben `13` zurück.

Die folgende Frage bringt zwischen 0 und 6 Punkte.
Du verlierst 3 Punkte für jede Zone , die nicht korrekt ausgefüllt ist.

Q2(t) /6	**Vervollständige die in der Funktion `Pyrdec(LIS,n)`.**

```
function Pyrdec(n,LIS) {
  dec ← 0
  v ← 1
  for r ← 0 to n-1 step 1 {
    v ← v * (r+1)

    dec ← dec + LIS[r] * v
  }
  return(dec)
}
```

Frage 3 – Der verlorene Palast Episode 3: Addieren.

Professor Zarbi hat alle Pläne entschlüsselt, auf denen kleine Scheibenpyramiden auf den Fliesen gezeichnet waren.
Er hat die Pläne abgeschrieben und dabei jede kleine Pyramide durch die Zahl ersetzt, die sie darstellt.
Nach langen Überlegungen und sorgfältigen Experimenten hat er endlich die Lösung gefunden...

Achtung! Nicht verwechseln!

In Episode 3 bestehen die Pyramiden wieder aus Fliesen, wie in Episode 1.
Aber in dieser Episode trägt jede Fliese eine Zahl (die in Episode 2 entschlüsselte).

Wege mit maximaler Summe.

Ab jetzt ist eine ***Zahlenpyramide*** eine Pyramide mit einer Zahl auf jeder Fliese.

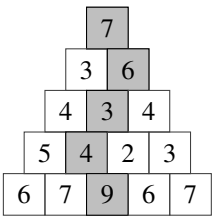
Zur Erinnerung: Ein ***Weg***

- beginnt an der Spitze,
- geht von einer Fliese zu einer anderen, die sie berührt,
- geht über genau eine Fliese in jeder Reihe,
- muss die letzte Reihe erreichen.

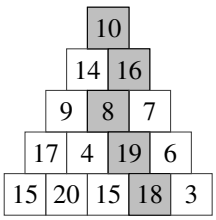
Die ***Punktzahl*** eines Weges ist die Summe der Zahlen auf den Fliesen dieses Weges.
Professor Zarbi hat entdeckt, dass der Weg mit der größten Punktzahl keine Falle auslöst.
Wenn mehrere Wege die gleiche maximale Punktzahl haben, löst keiner eine Falle aus.
Jeder Weg, der eine geringere als die maximale Punktzahl hat, löst eine Falle aus.

Beispiele: Hier sind einige Pyramiden **P5**.
Für jede ist ein Weg mit maximaler Punktzahl grau eingefärbt.
Die Punktzahl des Weges steht im Rechteck.

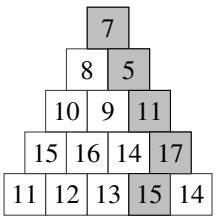
`7+6+3+4+9=29`



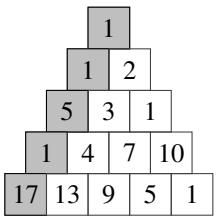
`10+16+8+19+18=71`



`7+5+11+17+15=55`



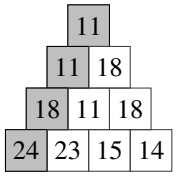
`1+1+5+1+17=25`



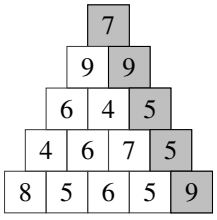
In der folgenden Aufgabe sind Pyramiden mit einer Zahl auf jeder Fliese gezeichnet. Zeichne in jeder Pyramide einen Weg mit maximaler Punktzahl (indem du die Zahlen einkreist oder die Fliesen mit Bleistift einfärbst).
Schreibe die Punktzahl des gezeichneten Weges in das Rechteck über der Pyramide.

Q3(a) /8	**Zeichne einen Weg mit maximaler Punktzahl und trage seine Punktzahl im Rechteck ein.**

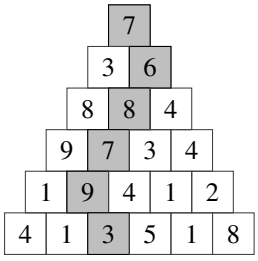
`64`



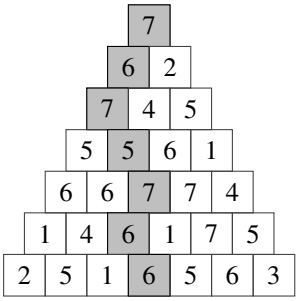
`35`



`40`



`44`



Automatisierung

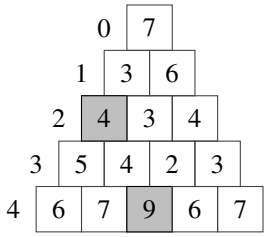
Professor Zarbi möchte die Berechnung der maximalen Punktzahl einer Zahlenpyramide automatisieren.
In seinen Programmen wird eine Pyramide **Pn** durch eine Liste `P` von `n` Teillisten dargestellt, nummeriert von `0` bis `n-1`.
Jede Teilliste enthält die Zahlen einer Reihe.
Die erste Teilliste `P[0]` enthält die Zahl an der Spitze der Pyramide.
Die letzte Teilliste `P[n-1]` enthält die `n` Zahlen der letzten Reihe.

Beispiel

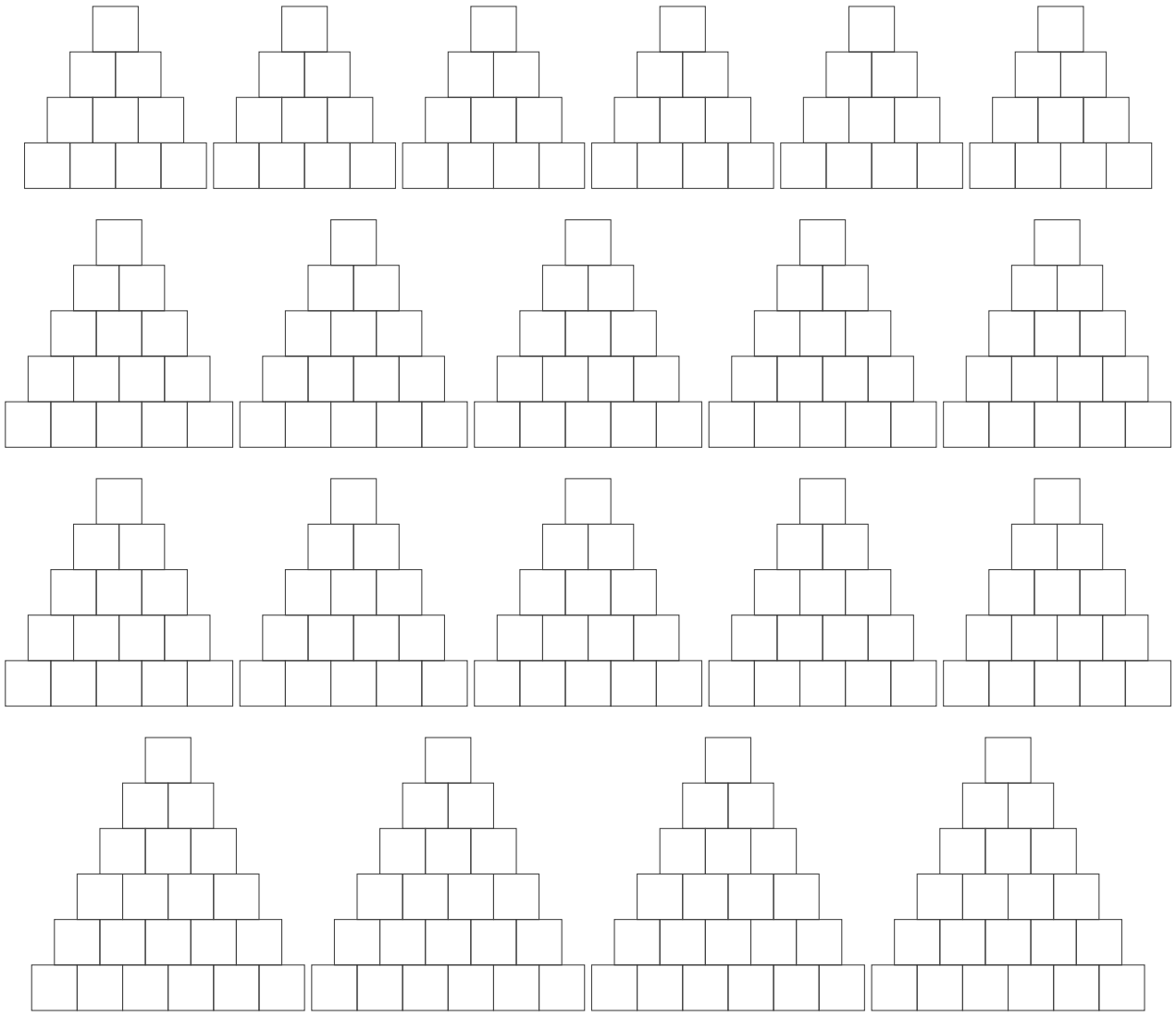
Die nebenstehende Pyramide **P5** wird mit ihren Reihennummern links angezeigt.
Für diese Pyramide:

```
n=5
P=[[7],[3,6],[4,3,4],[5,4,2,3],[6,7,9,6,7]]
P[0]=[7]
P[4]=[6,7,9,6,7]
```

`P[2][0]=4` und `P[4][2]=9` sind grau eingefärbt.



Pyramiden für deine Entwurfsarbeit.



Funktion `MaxScore(n,P)`
Diese Funktion verwendet 2 Parameter.

- `n`: die Anzahl der Reihen der Pyramide.
- `P`: eine Liste von Zahlenlisten, die die Pyramide darstellt, wie oben erklärt.

Die Funktion gibt die maximal mögliche Punktzahl für einen Weg in dieser Pyramide zurück.

Beispiel: `MaxScore(5,[[7],[3,6],[4,3,4],[5,4,2,3],[6,7,9,6,7]])` gibt `29` zurück (siehe das erste Beispiel auf der vorherigen Seite).

Für jede Fliese berechnet das Programm eine Punktzahl, die der maximalen Punktzahl eines Weges in der Teilpyramide mit dieser Fliese als Spitze entspricht.
Man kann die Punktzahl einer Fliese mithilfe der Punktzahlen der Fliesen berechnen, die sie in der nächsten Reihe berührt.
Die Punktzahl einer Fliese ersetzt in `P` die auf der Fliese geschriebene Zahl.

Beispiel

Während der Verarbeitung der linken Pyramide
ersetzt die Funktion die Werte in `P` durch die in der rechten Pyramide angezeigten.

Vor der Ausführung: `P=[[7],[3,6],[4,3,4],[5,4,2,3],[6,7,9,6,7]]`
Nach der Ausführung: `P=[[29],[20,22],[17,16,15],[12,13,11,10],[6,7,9,6,7]]`

Die folgende Aufgabe bringt zwischen 0 und 10 Punkte.
Du verlierst 2 Punkte für jedes Feld ____, das nicht korrekt ausgefüllt ist. Es gibt keine negativen Punkte.

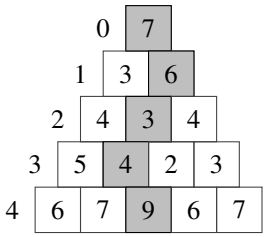
Q3(b) /10	**Vervollständige die ____ in der Funktion `MaxScore(n,P)`.**

```
function MaxScore(n,P) {
  for r ← n-2 to 0 step -1
  {
    for c ← 0 to r step 1
    {
      if P[r+1][c] > P[r+1][c+1]

        {P[r][c] ← P[r][c] + P[r+1][c]}
      else

        {P[r][c] ← P[r][c] + P[r+1][c+1]}
    }
  }
  return P[0][0]
}
```

Funktion `GoodPath(n,P)`
Was wirklich zählt, ist einen Weg mit maximaler Punktzahl zu finden, um keine Fallen auszulösen.
Ein Weg wird durch eine Liste dargestellt, die für jede Reihe die Nummer der Fliese enthält, über die der Weg in dieser Reihe führt.

Beispiel

Die nebenstehende Pyramide **P5** wird mit ihren Reihennummern links angezeigt.
Der grau eingefärbte Weg in der Pyramide wird durch die Liste `GP` der Fliesennummern in ihrer Reihe beschrieben.
In diesem Beispiel: `GP=[0,1,1,1,2]`
Der Weg führt nämlich über die Fliese `0` der Reihe `0`,
die Fliese `1` der Reihen `1`, `2` und `3`
und schließlich über die Fliese `2` der Reihe `4`.



Die Funktion `GoodPath(n,P)` verwendet die gleichen Parameter wie die Funktion `MaxScore(n,P)`.
Sie gibt eine Liste zurück, die einen Weg mit maximaler Punktzahl in `P` beschreibt.
Es genügt, das **`return`** der Funktion `MaxScore(n,P)` durch einige Codezeilen zu ersetzen.

Diese Aufgabe bringt zwischen 0 und 6 Punkte.
Du verlierst 2 Punkte für jedes Feld ____, das nicht korrekt ausgefüllt ist. Es gibt keine negativen Punkte.

Erinnerung: `GP←[0]*n` erstellt eine Liste von `n` Nullen.

Q3(c) /6	**Vervollständige die ____ im Code von `GoodPath`, der `return` in `MaxScore` ersetzt.**

```
c ← 0
GP ← [0]*n
for r ← 1 to n-1  n-1  1
{
  if  P[r][c] < P[r][c+1]

    { c ← c+1 }

  GP[ r ] ← c
}
return GP
```