

1 Aufgaben zur Spieltheorie

1.1 Beispiel Nimm-Spiel

1.1.1 Spielbeschreibung

Auf dem Tisch liegt ein Haufen von $\text{anzahl} = 5$ Streichhölzern. Jeder Spieler muss eine Anzahl zwischen 1 und 3 Streichhölzern vom Tisch nehmen. Wer nicht mehr ziehen kann (weil kein Streichholz mehr da ist) hat verloren.

Man kann das Spiel auch verallgemeinern:

Statt maximal 3 Streichhölzer kann man maximal k Streichhölzer nehmen.

1.1.2 Beispiel

Möglicher Spielverlauf:

(2, 3) Spieler1 zieht 2 Streichhölzer, also aktuelle Anzahl $5-2 = 3$

(1, 2) Spieler2 zieht 1 Streichholz, also aktuelle Anzahl $3-1 = 2$

(2, 0) Spieler1 zieht 2 Streichhölzer, also aktuelle Anzahl $2-2 = 0$

Spieler 2 ist am Zug und hat verloren, weil der Haufen leer ist.

1.1.3 Frage

Wie geht das Spiel aus, wenn der anziehende und der nachziehende Spieler jeweils den "besten Zug" machen ?

1.1.4 Fachbegriffe aus der mathematischen Spieltheorie

Nach dem Spielende von "Nimm-Spiel" gibt es für jeden Spieler in diesem sogenannten **Nullsummenspiel** (die Summe der Auszahlungen aller beteiligten Spieler ist Null) mit **vollständiger Information** (beide Spieler sind in jeder Phase des Spiels über die bestehende Situation genau informiert) eine **Auszahlung (payout)**:

1 : bei Gewinn,

-1 : bei Niederlage

bzw. falls es Spiele mit unentschieden (beim Nimm-Spiel gibt es kein unentschieden) gibt: Gewinn (1), Unentschieden (0), Niederlage (-1)

Diese Auszahlung hängt von den Spielzügen der einzelnen Spieler ab.

Für jeden Spieler in diesem Spiel gibt es eine **optimale Strategie** in dem Sinne, daß es einen Plan gibt, der eine höchstmögliche (maximale) Auszahlung sichert.

Um die optimale Strategie herauszufinden, kann man sämtliche möglichen Spiele in einem sogenannten **Spielbaum** (ähnlich dem Verzeichnisbaum des Window-Dateisystems) darstellen. Die Wurzel des Spielbaums ist die Ausgangsposition (Anfangsposition) des Spiels, also im oberen Beispiel (5).

Dann notiert man darunter baumförmig die möglichen weiteren Positionen. Dies wird so lange gemacht, bis man jeweils an eine Endposition kommt (d.h. wo entschieden werden kann, wie hoch die Auszahlung ist).

Zu den Positionen im Spielbaum sagt man auch **Knoten**. Ist ein Knoten eine Endposition, so ist dieser Knoten ein **Blatt**.

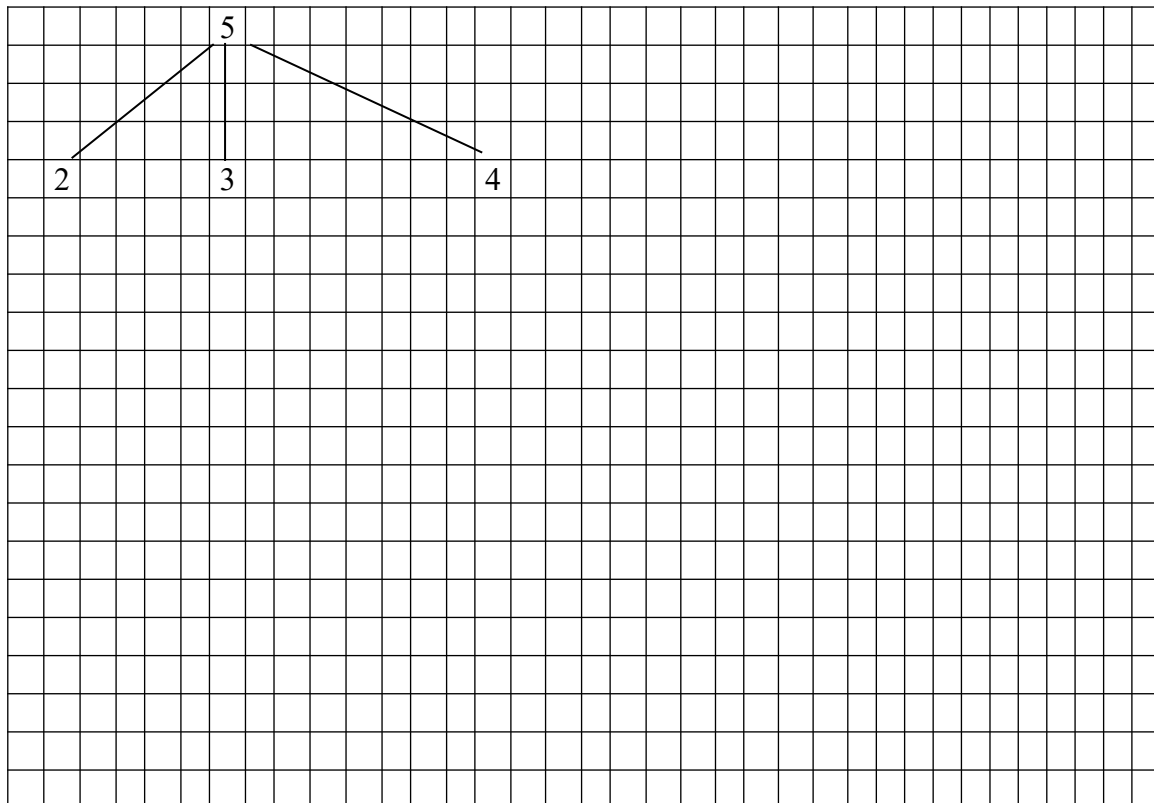
1.1.5 Spielbaum

Man kann alle möglichen Spielzüge in einem Spielbaum darstellen

Aufgabe

anzahl=5, k=3

Ergänzen Sie den angefangenen Spielbaum!



1.1.5.1 Aufgaben zum obigen Spielbaum

I)

- 1) Geben Sie die Auszahlung (payout) des anziehenden Spielers mit $\text{anzahl} = 0$ an.
- 2) Geben Sie die Auszahlung (payout) des anziehenden Spielers mit $\text{anzahl} = 1$ an.
- 3) Geben Sie die **möglichen** Auszahlungen (payout) des anziehenden Spielers mit $\text{anzahl} = 2$ an.
- 4) Angenommen, man würde die maximale Auszahlung (payout) $p_{1\max}$, $p_{2\max}$, $p_{3\max}$ aller möglichen Spielzüge (Nachfolgeknoten) des Gegners kennen.
Wie berechnet man daraus die maximale Auszahlung p_{\max} des Vorgängerknotens?
Tun Sie so, als ob Sie die maximale Auszahlungen des Gegners kennen und schreiben Sie diese an die Nachfolgeknoten. Überlegen Sie dann, wie man die maximale Auszahlung berechnet und schreiben diese an den Vorgängerknoten.

Füllen Sie dazu folgende Tabelle aus.

$p_{1\max}$	$p_{2\max}$	$p_{3\max}$	p_{\max}
-1	-1	-1	
-1	-1	1	
-1	1	-1	
-1	1	1	
1	-1	-1	
1	-1	1	
1	1	-1	
1	1	1	

Geben Sie die Funktion $p_{\max}(p_{1\max}, p_{2\max}, p_{3\max})$ an, die p_{\max} in Abhängigkeit von $p_{1\max}$, $p_{2\max}$ und $p_{3\max}$ berechnet.

Ergebnis:

$p_{\max} =$

II)

- 1) Schreiben Sie ein Programm, das p_{\max} des anziehenden Spielers in Abhängigkeit des Haufens berechnet (iterativ und rekursiv)
- 2) Schreiben Sie ein Programm, in dem ein Spieler gegen den Computer spielt (Nimm-Spiel). Der Computer gibt nach jedem Zug seine maximale Gewinnauszahlung aus und kann damit also schon lange **vor Spielende** dem Gegner mitteilen, dass der Computer gewonnen hat.

1.2 Würfelkippen

1.2.1 Spielbeschreibung

Beim Spielbeginn wird eine zu erreichende Summe (Sollsumme) festgelegt. Diese muß größer oder gleich 6 sein.

Dann wird ein Spielwürfel geworfen und die oben liegende Zahl als aktuelle Summe (Istsumme) notiert. Die beiden Spieler kippen nun abwechselnd den Würfel um eine der 4 Grundkanten und erhöhen die aktuelle Summe um die jeweils nach oben gelangte Augenzahl. (Das heißt, daß ein Spieler die vor dem Ziehen oben liegende Zahl und die Zahl auf der Gegenseite nicht ziehen kann !)

Erreicht ein Spieler durch seinen Zug diese Sollsumme, hat er gewonnen und der andere verloren. Ein Spieler darf durch seinen Zug auf keinen Fall die Sollsumme überschreiten. Kann er diese durch keinen Zug erreichen, ist das Spiel unentschieden.

Bemerkung:

Das Spiel kann dadurch noch modifiziert werden, daß die erste aktuelle Summe nicht gleich der ersten gezogenen Augenzahl ist, sondern eine beliebig vorgegebene sein kann.

1.2.2 Beispiel

Eine Partie des Spiels könnte zum Beispiel so ablaufen:

Sollsumme = 10

Erste gewürfelte Augenzahl = 2

Istsumme = 2

Dies kann man kurz darstellen als: (2 , 2)

Möglicher Spielverlauf:

(2 , 2)

(4 , 6) 4 gezogen, also Summe $4 + 2 = 6$

(2 , 8) 2 gezogen, also Summe $2 + 6 = 8$

(1 , 9) 1 gezogen, also Summe $1 + 8 = 9$

Das Spiel endet unentschieden, da der nachziehende Spieler nur noch die Zahlen zwischen 2 und 5 ziehen kann und damit die Sollsumme überschreiten würde.

1.2.3 Aufgaben

Erstellen Sie einen Spielbaum für

a) Sollsumme = 8 Erste gewürfelte Augenzahl = 4

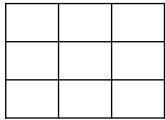
b) Sollsumme = 7 Erste gewürfelte Augenzahl = 1

c) Sollsumme = 15 Erste gewürfelte Augenzahl = 6

Geben Sie jeweils die maximale Auszahlung des anziehenden Spielers aus.

1.3 Tic Tac Toe

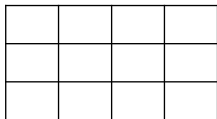
Wer zuerst 3 zusammenhängende Steine in einer Zeile, Spalte oder Diagonale hat, hat gewonnen.



Analysieren Sie das Spiel!

1.4 Zick Zack Two

Wer zuerst 4 zusammenhängende Steine in einer Zeile hat, oder 3 zusammenhängende in einer Spalte bzw. Diagonale, hat gewonnen.

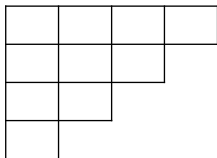


Analysieren Sie das Spiel!

1.5 Mini-Mühle

Wenn der Anziehende zuerst 3 zusammenhängende Steine waagrecht oder senkrecht hat, hat er gewonnen.

Wenn der Nachziehende zuerst 3 zusammenhängende Steine waagrecht, senkrecht oder diagonal hat, hat er gewonnen.



Analysieren Sie das Spiel!

1.6 Quinto

Analysieren Sie das Spiel (Beschreibung siehe Internet).

1.7 Solitaire (Brettspiel)

Analysieren Sie das Spiel (Beschreibung siehe Internet).

1.8 Ein Springer-Problem

Wie kann ein einzelner Springer auf dem 8x8-Schachbrett einen Weg finden, so dass er auf jedem Feld genau einmal vorbeikommt. Es gibt dabei verschiedene Varianten:

- 1) Es können nur geschlossene Wege gesucht werden.
- 2) Von einem festen Startpunkt aus darf der Weg irgendwo auf dem Schachbrett enden.

1.9 Ein anderes Springer-Problem

Ein Springer soll von einem beliebigen Feld eines 8x8-Schachbretts mit möglichst wenigen Zügen auf ein beliebiges anders Feld springen.

1.10 Das 8 - Damen-Problem

Es sollen insgesamt 8 Damen auf dem Schachbrett so verteilt werden, dass sich jeweils keine zwei Damen einander nach den Schachregeln schlagen können. Die Figurenfarbe wird dabei ignoriert, und es wird angenommen, dass jede Figur jede andere angreifen könnte.

So eine Stellung wird 8-unangreifbar genannt

In der Literatur wird dies oft als klassisches Beispiel für die Technik "Backtracking" verwendet.

Dies geht auch ohne Backtracking mit Rekindsatz 4 Version 2!

Eine mögliche Lösung:

							x
			x				
x							
		x					
					x		
	x						
						x	
				x			

entspricht -->

0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0

0: unbelegtes Element des Felds

1: durch eine Dame belegtes Element des Felds

1.11 Sudoku

Die restlichen Zellen eines vorbelegten 9x9-Feld (kurz Feld) sollen so mit Ziffern von 1 bis 9 besetzt werden (sudokuergänzbar), dass alle 9 Quadrate, alle Zeilen und Spalten jeweils unterschiedliche Ziffern enthalten.

1.12 Grundys Spiel

Grundys Spiel ist ein leicht zu verstehendes kombinatorisches Spiel. Ein Spielzug besteht einfach darin, einen Haufen von Spielsteinen auszuwählen und in ungleich große kleinere Haufen zu zerlegen. Wenn es keinen zerlegbaren Haufen mehr gibt, ist das Spiel beendet und der Spieler mit dem letzten Zug hat gewonnen. (genaue Beschreibung siehe Internet)

2 Aufgaben "Syntaxchecker"

2.1 Informationen

2.1.1 Scanner

Ein Scanner (lexikalischer Scanner, Tokenizer, Lexer) zerlegt eine Zeichenkette in zusammengehörigen Einheiten (Wörter, Atome, Token).

Liest der Scanner etwa die Zeichenfolge xyz , so könnte er die auf verschiedene Weisen interpretieren: ein Token xyz , zwei Token x , yz oder xy , z oder drei Token x , y , z

Um Eindeutigkeit zu bekommen, wird folgende Verabredung gemacht:

ein Token wird solange gelesen wird, bis es nicht mehr zu einem Token erweitert werden kann, d.h. bis das nächste Zeichen nicht mehr zu dem Token passt.

Dieses Vorgehen nennt man **maximum-munch**.

Beispiel:

$((x_1+y_1)*3.1415)$

zugehörige Token:

((x_1 + y_1) * 3.1415)

2.1.2 Parser

Ein Parser überprüft die Grammatik (und macht die Syntaxanalyse) einer Zeichenkette und bereitet diese für die Weiterverarbeitung entsprechend auf.

Beispiel:

$(x_1+y_1)*3.1415)$

Parser stellt fest, dass dies kein mathematischer Term ist (Eine schließende Klammer zu wenig).

Aufgaben

0)

Es soll nur ein Teil eines HTML-Syntax-Checks realisiert werden.

Eine Funktion `html_check1(...)` prüft die korrekte Verwendung von Tag-Klammern. Sie erhält eine Zeichenkette, die den zu prüfenden HTML-Quelltext enthält und liefert den Wert 0 wenn alles in Ordnung ist. Falls ein Fehler gefunden wird, liefert die Methode die Position im Text, an der der Fehler gefunden wurde.

Zu prüfende Regel: auf eine spitze öffnende Klammer folgt immer eine spitze schließende Klammer, geschachtelte Klammerpaare sind also nicht erlaubt. Die Funktion erkennt das erste Auftreten einer falschen oder fehlenden Klammer und gibt die Position des Fehlers zurück. Die Inhalte der Tags - also das was zwischen den Klammern steht - werden nicht geprüft.

Beispiele ($x_1 x_2, x_3, \dots$ sind Zeichen)

Beispiel 1: `< x1x2>x1x2x3<x7>` (korrekt, Funktion liefert Rückgabe 0)

Beispiel 2: `<>x8<x4>` (korrekt, Funktion liefert Rückgabe 0)

Beispiel 3: `<>` (korrekt, Funktion liefert Rückgabe 0)

Beispiel 4: `<x1<x2x3>` (zweite öffnende Klammer an Pos. 3, Rückgabe 3)

Beispiel 5: `<x1>x2x3<x4` (schließende Klammer fehlt am Ende, Rückgabe 8)

Beispiel 6: `x1>x2x3<x4>` (schließende Klammer schon am Anfang, Rückg. 2)

Erstellen Sie eine Lösung mit und ohne Rekursion !

1)

Erstellen Sie eine rekursive Funktion, die feststellt, ob eine Zeichenkette eine positive Fließkommazahl ist. (Geben Sie die Regeln dazu an)

Beispiele für Fließkommazahlen:

3.14

0.2718

2)

Erstellen Sie eine rekursive Funktion, die feststellt, ob eine Zeichenkette ein regulärer Ausdruck der Form

$(a|b)^*(cb)^*$

ist.

3) Parser

$A = \{a,b,c,+ \}$ ist das Alphabet. Terme sind spezielle Zeichenketten über X , die das Rechenzeichen $+$ enthalten dürfen.

Beispiele für Terme über A :

a

b

c

$(a+b)$

$((c+b)+a)$

...

Schreiben Sie ein Programm (Parser), das von einer Zeichenkette über A feststellt, ob diese ein Term ist oder nicht.

4) Parser

$A = \{a, b, c, +, =, \wedge, \vee\}$ ist das Alphabet. Formeln sind spezielle Zeichenketten über X, die das Rechenzeichen +, die logischen Zeichen \wedge, \vee und das Gleichheitszeichen = enthalten dürfen.

Beispiele für Formeln über A:

(a = b)

((b+c) = (c+a))

((b=c) \wedge (b+c)) \vee (((c+a)+b) = a))

...

Schreiben Sie ein Programm (Parser), das von einer Zeichenkette über A feststellt, ob diese eine Formel ist oder nicht.

5)

$A = \{a,b,c,1,2,3,+ \}$ ist das Alphabet. Terme sind spezielle Zeichenketten über X, die das Rechenzeichen + enthalten dürfen.

Atomare Terme (d.h. die nicht aus Rechenzeichen zusammengesetzt sind) haben die Form wie Variablen einer Programmiersprache (hier mit maximaler Länge 5) und dürfen nicht mit einer Zahl beginnen und dürfen nicht aus lauter Ziffern bestehen

Beispiele für atomare Terme über A:

abacb

b1ca

c

a12

...

Nichtatomare Terme werden wie bei Aufgabe 3) gebildet.

Eine Zeichenkette wird über Tastatur eingegeben, wobei die "zusammengehörigen" Elemente durch ein Leerzeichen getrennt werden, wie z.B:

((a1 + b23) + (c2 + b1))

a)

Schreiben Sie ein Programm (Einfach-Scanner), das diese Token in einem Array speichert.

Beispielsweise wird die folgende Zeichenkette

((a1 + b23) + (c2 + b1))

in dem folgenden Feld abgespeichert.

((a1	+	b23)	+	(c2	+	b1))
---	---	----	---	-----	---	---	---	----	---	----	---	---

b) Schreiben Sie ein Programm (Parser), das von einer in die Grundelemente zerlegten Zeichenkette feststellt, ob diese ein Term ist.

6)

$A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, - \}$ ist das Alphabet. Terme sind spezielle Zeichenketten über X, die das Rechenzeichen - enthalten dürfen.

Beispiele für Terme über A:

3 --> Wert 3

7 --> Wert 7

(3-7) --> Wert -4

((2-5)-6) --> Wert -9

((2(-)-))) --> Wert #

...

Schreiben Sie ein Programm, das von einer Zeichenkette über A den Wert ermittelt (falls diese Zeichenkette ein Term ist).

Falls diese Zeichenkette kein Term ist soll nicht -1 zurückgeliefert werden (wie in der Präsentation), sondern das Zeichen #

Um diese programmtechnisch zu realisieren, kann man eine Struktur verwenden, in der neben Zahlen auch noch das Zeichen # vorkommt.

7) Scanner

In Aufgabe 5a) werden die Token bei der Eingabe mit Hilfe eines freundlichen Users durch ein Leerzeichen getrennt. Der Scanner hat dann nicht mehr viel zu tun ...

Wenn der User nicht mehr so freundlich ist und die Token durch Leerzeichen trennt, muß dies die in den Scanner implementierte Intelligenz machen.

Grobe Beschreibung des Algorithmus:

i1) Man schreibt Funktionen, wie

boolean ist_Zahl(char string[])

boolean ist_Rechenzeichen(char string[])

usw.

Diese ermitteln, ob ein bestimmter String eine Zahl, ein Rechenzeichen, usw. ist.

i2) Mit diesen Funktionen wird nun ein array aus Zeichen untersucht.

Zuerst wird das längste Teilfeld (beginnend beim Index 0) des array ermittelt, für das eine der obigen ist ... Funktionen den Wert true zurückgibt.

Angenommen dieses Teilfeld hat die Länge 5.

Dann wird das längste Teilfeld (beginnend beim Index 5) des array ermittelt, für das eine der obigen ist... Funktionen den Wert true zurückgibt.

Das wird so lange gemacht, bis das Ende des Array erreicht wird oder kein Token ermittelt werden kann.

8) Mini-C

Es soll ein Syntaxchecker einer Mini-C-Programmiersprache entwickelt werden, d.h. ein Programm, das nachprüft, ob ein Mini-C-Programm syntaktisch korrekt ist.

i1) Variablen

bestehen aus Zeichenfolgen mit maximaler Länge 2, deren Zeichen Kleinbuchstaben, aber keine Umlaute sein müssen.

Beispiele: x, ba

i2) Zahlen

müssen positiv sein und dürfen Gleitkommazahlen sein..

Beispiele: 8, 2.718

i3) Rechenzeichen

Rechenzeichen sind +, -, *, /

i4) logische Zeichen

logische Zeichen sind &&, ||, ~

i10) arithmetische Terme

Beispiele: $((x+zy)-3.14)$

i11) Formeln

Beispiele:

$((a+b) == ((2.78-x)*y))$

i12) einfache Anweisungen

haben die Form:

Variable = Term;

Beispiele: $rs = ((x+zy)-3.14)$

i12) if-else-Anweisungen

haben die Form:

if{ Formel Anweisung} else {Anweisung}

i13) Anweisungen

sind entweder einfache Anweisungen oder (verschachtelte) if-else-Anweisungen

3 Weitere Aufgaben

3.1 wunderbare Zahlen oder wie ein "Universum" expandiert

3.1.1 Definition

a)

Es sollen die "wunderbaren Zahlen" durch Expansion entwickelt werden:

(analog dem Universum, das sich aus dem Urknall immer weiter expandiert hat)

Aus den bestehenden Zahlen werden immer wieder nach einer bestimmten Vorschrift neue gebildet und an das Dateiende angefügt (aber nur wenn sie nicht darin schon enthalten sind).

Vorschrift:

Nimm jeweils 2 verschiedene Zahlen, bilde deren Summe und füge sie an das Dateiende an, aber nur, wenn diese nicht schon in der Datei enthalten sind.

Man beginnt z.B. nach dem Urknall mit den Atomen 3 und 5:

{3 ; 5}

1. Expansion: {3 ; 5 ; 8}

2. Expansion: {3 ; 5 ; 8 ; 11 ; 13}

3. Expansion: {3 ; 5 ; 8 ; 11 ; 13 ; 14 ; 16 ; 18 ; 19 ; 21 ; 24}

...

Wenn man "unendlich" oft expandiert, bekommt man die Menge der wunderbaren Zahlen.

Bemerkung:

Man könnte diese "wunderbaren Zahlen" auch in ein (genügend großes) Feld oder in eine verkettete Liste oder in eine MySQL-Datenbank schreiben.

b)

Wenn genügend oft expandiert wird, ist es möglich, daß ab einer bestimmten Zahl die Zahlen keine Lücken mehr haben. Es könnte z.B. sein, daß ab der Zahl 53 die neu hinzukommenden Zahlen 54, 55, 56, 57, 58, 59, ... sind.

Dann könnte man diese Zahl 54 die Barkon-Konstante nennen, also wäre dann:

$\text{Barkon}(3,5) = 53$

Bestimmen Sie die Barkon-Konstante bei einem Universum, das mit einem anderen Urknall beginnt z.B. mit 13 und 17, also $\text{Barkon}(13,17)$.

3.1.2 Andere, selbst gebastelte wunderbaren Zahlen

Man kann andere, selbst erzeugte Zahlenmengen basteln:

V1) Es können andere Anfangszahlen (und eine andere Anzahl) benutzt werden.

V2) Es können andere Vorschriften benutzt werden, wie z.B:

$(x, y) \rightarrow x + y$, falls $x \neq y$

$(x, y) \rightarrow |x - y|$, falls $x \neq y$

$(x, y) \rightarrow x + y$, falls $x \neq y$ und nicht x und y beide gerade Zahlen

Man könnte sich folgende Fragen überlegen:

F1) Gibt es eine Zahl z , aber der alle darauf folgenden Zahlen $z+1, z+2, z+3, \dots$ (ohne Lücke) in der Menge der wunderbaren Zahlen vorkommen ?

F2) Kann man die Anfangszahlen so vorbelegen (und die Vorschrift so gestalten) , daß dies gelingt ?

F3) usw.

3.2 Anzahl Aufrufe bei den wunderbaren Zahlen

$$X = \{1; 2; 3; 4; 5 \dots\}$$

Regelkurznotation: $(r \in X, s \in X)$

$$\begin{array}{ccc} \emptyset & \emptyset & \{r; s\} \\ \text{---} & \text{-----} & \text{-----} \quad \text{mit } r \neq s \\ 3 & 5 & r + s \end{array}$$

Die Funktion $e(n)$ soll berechnen, ob eine Zahl n aus X zu der induktiv definierten Menge gehört oder nicht.

Wie oft ruft die Funktion $e(n)$ sich selbst auf?

3.3 Rätsel

$$X = \{1; 2; 3; 4; 5 \dots\}$$

Regelkurznotation: $(r \in X, s \in X)$

$$\begin{array}{cccc} \emptyset & \emptyset & \{r; s\} & \{z\} \\ \text{---} & \text{-----} & \text{-----} & \text{-----} \quad \text{mit } \sqrt{z} \text{ ganzzahlig} \\ 5 & 7 & r + s & \sqrt{z} \end{array}$$

3.4 Rätsel

a)

Erzeugen Sie eine Zahlenfolge, in der 2, dann 10 und zum Schluss 14 vorkommt.
 $\dots, 2, \dots, 10, \dots, 14$

Es müssen dabei die folgenden Bedingungen gelten:

- Keine Zahl darf doppelt vorkommen
- Jede Zahl muss kleiner oder gleich 64 sein.
- jede Zahl bekommt man aus der vorhergehenden Zahl durch Addition von 5 bzw. 7 oder durch Wurzelziehen, wobei die berechnete Wurzel eine natürliche Zahl sein muss (wie z.B. Wurzel aus 36)

Beispiel für eine Reihenfolge:

$$5, 5+5, 5+5+5, 5+5+5+7, 5+5+5+7+7, 5+5+5+7+7+7, \sqrt{5+5+5+7+7+7}$$

Zusammengefasst:

$$5, 10, 15, 22, 29, 36, 6,$$

Beispiel:

$$5, 10, 15, 22, 29, 36, 6, 11, 16, 4, 9, 3, 10, 15, 20, 25, 30, 35, 42, 49, 7, 14$$

Tipp:

Erzeugen Sie nacheinander die Folgen von Listen, wobei eine Folge aus der vorhergehenden produziert wird:

$$(5), (7)$$

$$(5, 10), (5, 12), (7, 12), (7, 14)$$

$$(5, 10, 15), (5, 10, 17), (5, 12, 17), (5, 12, 19), (7, 12, 17), (7, 12, 19), (7, 14, 19), (7, 14, 21)$$

....

b)

Wie viele Zahlenfolgen mit obiger Eigenschaft gibt es ?

c)

Erzeugen Sie eine Zahlenfolge mit minimaler Länge (Anzahl von Zahlen).

d)

Erzeugen Sie eine Zahlenfolge mit maximaler Länge (Anzahl von Zahlen).

3.5 Zahlen

Schreiben Sie ein Programm, das Ihnen alle n - stelligen Zahlen (z.B. $n = 6$) ausgibt, die mit den Ziffern 1,2,3 gebildet werden können.

3.6 Zahlen verteilen

Die Zahlen der Menge $\{1, \dots, p\}$ sollen auf q Zellen verteilt werden, so daß sich in den q Zellen keine zwei gleichen Zahlen befinden. Es sollen alle Möglichkeiten angegeben werden.

Beispiel:

$p = 4$ und $q = 3$

123; 213; 312; 412; 132; 231; 321; 421; 124; 234; 314; 423; 142; 243; 341; 432;
134; 214; 324; 413; 143; 241; 342; 431

3.7 Das magische Quadrat

Ein teilweise (oder völlig) unbesetztes 3×3 -Feld (kurz Feld) soll so mit verschiedenen Ziffern von 1 bis 9 besetzt (magisch ergänzt) werden, dass es magisch wird, d.h alle Ziffern müssen verschieden und die jeweiligen Zeilensummen, Spaltensummen, Hauptdiagonalensummen müssen gleich groß sein.

3.8 Erfüllbarkeitsproblem der Aussagenlogik

Beschreibung:

Gegeben ist eine boolesche Formel F (mit n Variablen x_1, \dots, x_n) in konjunktiver Normalform, also eine UND-Verknüpfung von Klauseln, wie z.B:

$$F = x_1 + x_2 + x_3' \cdot x_1' \cdot x_2 x_3 \cdot x_1 x_4$$

und die booleschen Variablen die Werte 0 (für falsch) und 1 (für wahr) annehmen.

Ist F erfüllbar, d.h. es gibt eine Belegung der Variablen mit Wahrheitswerten, so daß die ausgewertete Formel F den Wert 1 erhält?

Bemerkung:

Z.B. bedeutet x_3' eine negierte boolesche Variable

+ bedeutet das logische ODER und \cdot das logische UND

Beispiel:

$$F = x_1 + x_2 + x_3' \cdot x_1' \cdot x_2 x_3 \cdot x_1' x_4$$

Man beginnt mit der leeren Belegung ϵ .

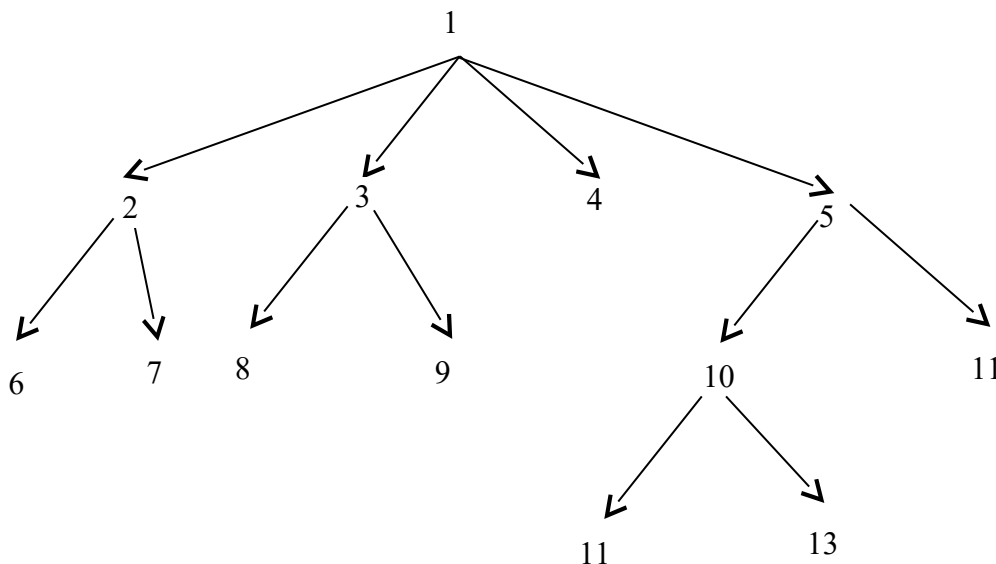
Die Belegungen können von $(0,0,0,0)$ bis $(1,1,1,1)$ gehen.

3.9 Tiefe eines Baums bestimmen

Die Tiefe eines Baums ist die Länges des längsten Pfades von der Wurzel zu einem Blatt.

Per Definition soll ein Graph mit genau einem Knoten die Tiefe 1 haben.

Beispiel: Tiefe 4



3.10 Kürzester Abstand zweier Knoten in einem Graph

Voraussetzungen:

$V(G) = \{ n_1, \dots, n_n \}$ ist die Knotenmenge des Graphen G .

Je zwei benachbarte Knoten i und j haben einen bestimmten Abstand $a(i, j)$ voneinander, der im Beispiel durch eine Zahl an der Kante dargestellt wird.

Mit $[k_1, \dots, k_n, k]$ wird ein Pfad in einem Graphen bezeichnet, wobei die k_1, \dots, k_n Elemente aus der Knotenmenge des Graphen G alle benachbart und paarweise verschieden sind.

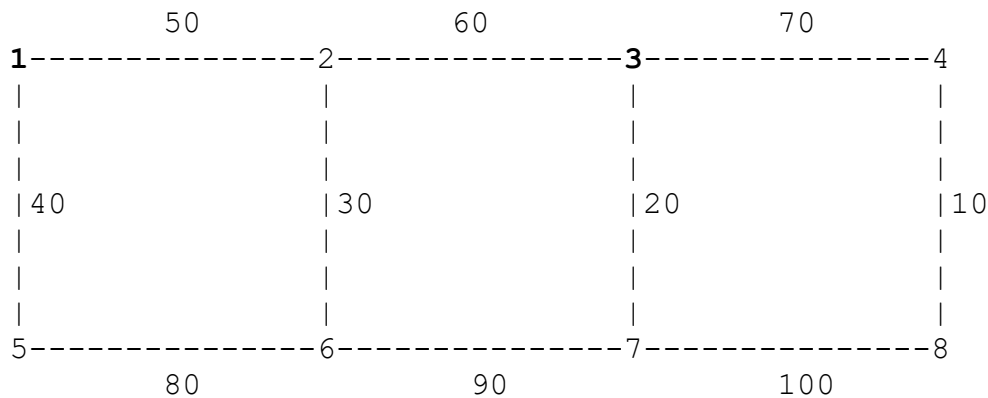
Gesucht ist der kürzeste Abstand $d([k_1, \dots, k_n, k])$ zweier Knoten k_n und k des Graphen, wobei dieser Weg nicht durch einen der Knoten $\{ k_1, \dots, k_{n-1} \}$ gehen darf.

Beispiel:

Die Knotenmenge des Graphen ist $\{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

Die Zahlen an den Kanten bedeuten die Abstände zwischen 2 Knoten.

Der kürzeste Abstand zwischen den Knoten 1 und 3 beträgt $d([1,3]) = 50 + 60 = 110$



Die Entfernungen kann man in einer Matrix (zweidimensionales Feld) darstellen:

	1	2	3	4	5	6	7	8
1	0	50	-1	-1	40	-1	-1	-1
2	50	0	60	-1	-1	30	-1	-1
3	-1	60	0	70	-1	-1	20	-1
4	-1	-1	70	0	-1	-1	-1	10
5	40	-1	-1	-1	0	80	-1	-1
6	-1	30	-1	-1	80	0	90	-1
7	-1	-1	20	-1	-1	90	0	100
8	-1	-1	-1	10	-1	-1	100	0

3.11 Kürzeste Rundreise in einem Graph

Voraussetzungen:

$V(G) = \{ n_1, \dots, n_n \}$ ist die Knotenmenge des Graphen G .

Je zwei benachbarte Knoten i und j haben einen bestimmten Abstand $a(i, j)$ voneinander, der im Beispiel durch eine Zahl an der Kante dargestellt wird.

Mit $[k_1, \dots, k_n, k]$ wird ein Pfad in einem Graphen bezeichnet, wobei die k_1, \dots, k_n Elemente aus der Knotenmenge des Graphen G alle benachbart und paarweise verschieden sind.

Gesucht ist der kürzeste Abstand $d([k_1, \dots, k_n, k])$ zweier Knoten k_n und k des Graphen, wobei dieser Weg nicht durch einen der Knoten $\{ k_1, \dots, k_{n-1} \}$ gehen darf und alle Knoten des Graphen genau einmal besucht werden, außer der Anfangsknoten ist der Endknoten.

Ein Pfad, der alle Knoten genau einmal besucht, wird vollständiger Pfad genannt.

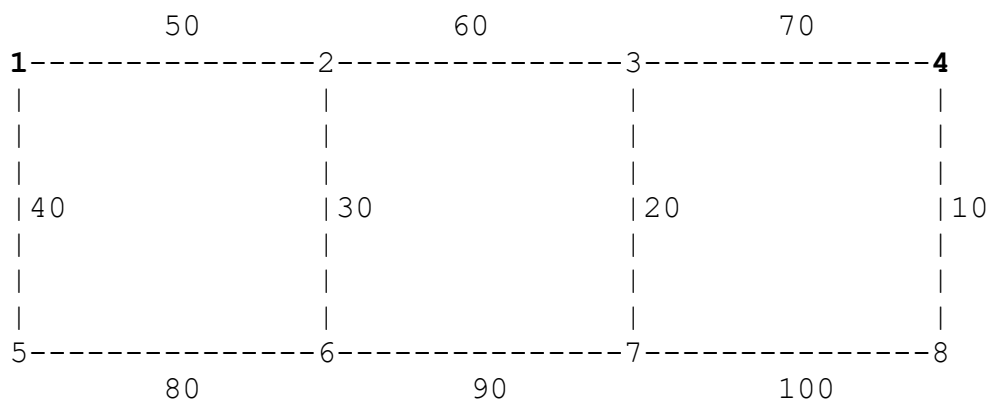
Ein Pfad, der alle Knoten genau einmal besucht und dessen Anfangsknoten gleich dem Endknoten ist, wird vollständiger Kreis genannt.

Beispiel:

Die Knotenmenge des Graphen ist $\{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

Die Zahlen an den Kanten bedeuten die Abstände zwischen 2 Knoten.

Der kürzeste Abstand zwischen den Knoten 1 und 3 beträgt $d([1,3]) = 50 + 60 = 110$



3.12 Schiebepuzzle

Rätsel der Woche Spiegelonline 20.5.18

Gegeben ist folgendes Schiebepuzzle, das aus 5 nummerierten Spielsteinen besteht.

2		3
1	5	4

Die möglichen erste Züge könnte z.B. so aussehen:

	2	3
1	5	4

2	5	3
1		4

2	3	
1	5	4

Wie viele Züge braucht man minimal, um die 2 mit der 1 zu vertauschen, d.h. bis folgender Stand erreicht ist (wobei die Fragezeichen bedeuten, dass dort die Steine 3, 4, 5 in beliebiger Anordnung stehen)

1		?
2	?	?

3.13 Rekursion

Implementieren Sie die folgende rekursive Funktion:

$$f(n) = f(f(n-1)-9)+1 \quad \text{für } n > 1$$

$$f(1) = 10$$