SS2025 FDS2 Ü01

Author: Tim Peko **Aufwand in h**: 8h

Beispiel 2

Lösungsansatz

Wir wollen einen Algorithmus entwickeln, der beide Stapel leert. Die einfachste Lösung für unseren "Computer" Spieler ist, dass wir immer gerade Werte auf beiden Stapeln anstreben.

Unser Endziel ist der Zustand

$$(0, 0)$$

Wir können diesen erreichen, indem mindestens einer der beiden Werte einen Betrag von 0 hat. Bedeutet:

$$(0, 1), (1, 1), (1, 0)$$

Dadurch dass der *Menschliche* Spieler immer mindestens einen Stein wegnehmen muss, lässt sich dieser Zustand herstellen, indem der *Computer* Spieler sicherstellt, dass beide Stapel immer eine gerade Zahl haben.

Gehen wir bei jedem Zug von diesen geraden Stapeln aus. Der *Menschliche* Spieler muss jeden Zug einen Stein wegnehmen, bedeutet mindestens einer der beiden Stapeln stellt eine ungerade Zahl dar. Er kann damit niemals auf den $(0, 0)$ Zustand kommen, da dieser zwei gerade Zahlen beinhaltet.

Wir halten fest, dass der *Menschliche* Spieler seinen Zug immer mit einem oder zwei ungeraden Stapeln beendet. Der *Computer* Spieler kann aus Zustand:

$$(2\mathbb{Z}, 2\mathbb{Z} + 1)|(2\mathbb{Z} + 1, 2\mathbb{Z} + 1)|(2\mathbb{Z} + 1, 2\mathbb{Z})$$

immer einen geraden Zustand:

$$(2\mathbb{Z}, 2\mathbb{Z})$$

herstellen, indem er entweder:

- $(0, -1)$
- $(-1, -1)$
- $(-1, 0)$

addiert.

Dies wiederholen wir solange, bis der *Computer* Spieler am Zielzustand $(0, 0)$, der ebenfalls $(2\mathbb{Z}, 2\mathbb{Z})$ entspricht, angelagt ist.

Der *Menschlische* Spieler hat keine Chance mehr zu gewinnen, sobald der *Computer* seinen Zug mit $(2\mathbb{Z}, 2\mathbb{Z})$ beenden kann. Der *Menschliche* Spieler kann nur gewinnen, indem er seine Züge allseits mit diesem Zustand beenden kann, denn dann hat der *Computer* keine Chance, seinen Zug mit $(2\mathbb{Z}, 2\mathbb{Z})$ zu beenden.

Begeht der *Menschliche* Spieler jedoch nur einen einzigen Fehler (Zustand der ein $2\mathbb{Z} + 1$) beinhaltet, kann der *Computer* Spieler diese Öffnung nutzen und so seinen eigenen $(2\mathbb{Z}, 2\mathbb{Z})$ Zustand etablieren.

Bedeutet also konkret für den *Computer* Zug:

Current State	Modification
$(2\mathbb{Z} > 0, 2\mathbb{Z} > 0)$	$(-1, -1)$
$(2\mathbb{Z} > 0, 0)$	$(-1, 0)$
$(0, 2\mathbb{Z} > 0)$	$(0, -1)$
$(2\mathbb{Z} + 1, 2\mathbb{Z})$	$(-1, 0)$
$(2\mathbb{Z} + 1, 2\mathbb{Z} + 1)$	$(-1, -1)$
$(2\mathbb{Z}, 2\mathbb{Z} + 1)$	$(0, -1)$

Testfälle

Folgende Testfälle gibt es:

1. User beendet Zug mit ungeraden Werten
   - Resultat: Computer gewinnt
2. User beendet Zug nie mit ungeraden Werten
   - Resultat: Mensch gewinnt
3. Stapel M ist leer, N ist ungerade und Mensch beginnt
   - Resultat: Mensch gewinnt
4. Stapel N ist leer, M ist gerade und Mensch beginnt
   - Resultat: Computer beginnt

Der Konsolen Output der einzelnen Testcases wird hier per angehängt.

Case 1

```
Enter the size of m stack
12d
That was no valid number
Enter the size of m stack
4
Enter the size of n stack
3
Would you like to take the first turn (0/1):
3
You have to enter either '0' or '1'
Would you like to take the first turn (0/1):
1
-----Current state-----
m -> 4  n -> 3
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
3
That action is not possible
1) Take from one stack
2) Take from both stacks
2
-----Current state-----
m -> 3  n -> 2
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 2  n -> 2
```

```
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
1
-----Current state-----
m -> 1  n -> 2
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 0  n -> 2
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
1
That stack choice is not valid
1) Take from M stack
2) Take from N stack
2
-----Current state-----
m -> 0  n -> 1
----------------------
The computer is now calculating next step...
The computer has made its turn
The computer has won! Better luck next time!
Thanks for playing
```

Result: success

Case 2

```
Enter the size of m stack
4
Enter the size of n stack
6
Would you like to take the first turn (0/1):
0
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 3  n -> 5
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
2
-----Current state-----
m -> 2  n -> 4
```

```
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 1  n -> 3
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
2
-----Current state-----
m -> 0  n -> 2
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 0  n -> 1
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
2
-----Current state-----
m -> 0  n -> 0
----------------------
You have won! Congratulations!
Thanks for playing
```

Result: success

Case 3

```
Enter the size of m stack
0
Enter the size of n stack
5
Would you like to take the first turn (0/1):
1
-----Current state-----
m -> 0  n -> 5
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
2
That action is not possible
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
2
-----Current state-----
```

```
m -> 0  n -> 4
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 0  n -> 3
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
2
-----Current state-----
m -> 0  n -> 2
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 0  n -> 1
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
2
-----Current state-----
m -> 0  n -> 0
----------------------
You have won! Congratulations!
Thanks for playing
```

Result: success

Case 4

```
Enter the size of m stack
4
Enter the size of n stack
0
Would you like to take the first turn (0/1):
1
-----Current state-----
m -> 4  n -> 0
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
1
```

```
-----Current state-----
m -> 3  n -> 0
----------------------
The computer is now calculating next step...
The computer has made its turn
-----Current state-----
m -> 2  n -> 0
----------------------
It's your turn. Select one of the following:
1) Take from one stack
2) Take from both stacks
1
Select the stack you want to take from:
1) Take from M stack
2) Take from N stack
1
-----Current state-----
m -> 1  n -> 0
----------------------
The computer is now calculating next step...
The computer has made its turn
The computer has won! Better luck next time!
Thanks for playing
```

Result: success

Beispiel 2

Lösungsansatz

Computer Züge

Unsere Aufgabe besteht nicht darin, einen *optimalen* Spielzug zu generieren. Wir müssen lediglich eine Strategie entwickeln, bei der der *Computer* in der Lage ist, zu gewinnen. Es soll dem *menschlichen* Spieler ermöglicht werden, eine Runde zu spielen.

Folgende Observationen für das Spielverhalten:

- Es kann *immer* gewonnen werden, wenn zu Beginn des Zuges nur noch eine Reihe vorhanden ist.
- Es kann *immer* verloren werden, wenn wir einen Zug mit mit nur einer Reihe beenden.

Wir müssen daher verhindern, dass wir in die letztere Situation geraten und erkennen, dass wir uns in der ersteren Situation befinden, um das Spiel zu gewinnen.

Das kann auf folgende Regeln vereinfacht werden:

$$C = \sum_{i=0}^{n} \min\{R_i, 1\}$$

C... Die Anzahl an nicht leeren Reihen

n... Die Anzahl an Reihen in unserem Spiel

Cases

- *input state* => *action* => *output state*
- $C \in 2\mathbb{Z}$ => nimm einen Stein von der größten Reihe => $\begin{cases} C \in 2\mathbb{Z} \text{ if } R_{\max} > 1 \\ C \in 2\mathbb{Z}+1 \text{ if } R_{\max} = 1 \end{cases}$
- $C \in 2\mathbb{Z} + 1$ => nimm alle Steine von der kleinsten Reihe => $C \in 2\mathbb{Z}$

Wobei $C \in 2\mathbb{Z}$ ein *favorabler* Output Zustand ist, $C \in 2\mathbb{Z} + 1$ ein *nicht favorabler* Zustand ist.

- *Favorabel* definiert einen Zustand, aus dem wir nicht direkt verlieren können.

- *Nicht Favorabel* definiert einen Zustand, aus dem eine direkte Niderlage möglich ist.

Datei einlesen
Die einzulesene Datei folgt folgendem Format:

$$<\text{FÄNGT\_MENSCH\_AN:0|1}>;R_0;R_1;...;R_n$$

Testfälle

Happy Path
Input file:

```
1;4;5;7;1;2
```

Output:

```
------Current state------
0->4|1->5|2->7|3->1|4->2
------------------------
It's your turn. Select the row you want to take from:
2
Select the amount of stones you want to take:
7
------Current state------
0->4|1->5|2->0|3->1|4->2
------------------------
Computer made its turn.
------Current state------
0->4|1->4|2->0|3->1|4->2
------------------------
It's your turn. Select the row you want to take from:
3
Select the amount of stones you want to take:
3
You cannot take that many stones, take a different amount.
Select the amount of stones you want to take:
0
You cannot take that many stones, take a different amount.
Select the amount of stones you want to take:
1
------Current state------
0->4|1->4|2->0|3->0|4->2
------------------------
Computer made its turn.
------Current state------
0->4|1->4|2->0|3->0|4->0
------------------------
It's your turn. Select the row you want to take from:
1
Select the amount of stones you want to take:
3
------Current state------
0->4|1->1|2->0|3->0|4->0
------------------------
Computer made its turn.
------Current state------
0->3|1->1|2->0|3->0|4->0
```

```
------------------------
It's your turn. Select the row you want to take from:
0
Select the amount of stones you want to take:
2
------Current state------
0->1|1->1|2->0|3->0|4->0
------------------------
Computer made its turn.
------Current state------
0->0|1->1|2->0|3->0|4->0
------------------------
It's your turn. Select the row you want to take from:
1
Select the amount of stones you want to take:
1
You have won! Congratulations!
Thanks for playing
```

Result: success

Ungültiges Format: kein delimiter

Input file:

```
1 3;5;7;9
```

Output:

```
invalid gamestate.csv format
```

Result: success

Ungültiges Format: nicht-numerisch

Input file:

```
1;a;3;5
```

Output:

```
------Current state------

------------------------
It's your turn. Select the row you want to take from:
```

Result: failed

After fixing

Output:

```
gamestate.csv does not contain valid rows
```

Result: success

Leere Datei

Input file:

Output:

```
invalid gamestate.csv format
```

Result: success

Beispiel 3

Lösungsansatz

Es wird der Input zuerst nach links abgesucht ausgehend von der Position i. Dabei wird das erste Auftreten einer öffnenten Klammer "`(`" notiert oder - falls nicht gefunden - der Anfang des Inputs.

Als zweiter Schritt wird der Input nach rechts abgesucht ausgehend von der Position i. Es wird die Anzahl an Auftreten der öffnenten Klammer "`(`" C_o mitgespeichert. Wir haben das Ende unseres Subsekmentes erreicht, sobald die Anzahl der aufgetretenen geschlossenen Klammern "`)`" $C_g > C_o$. Falls nicht gefunden, entspricht das dem Ende des Inputs.

Testfälle

Normal Happy Path

p_in = `.(((((..(((...)))..((..)))))...))).`

i = 6

out = `(..(((...)))..((..)))`

Result: success

Minimaler Fall

p_in = `.`

i = 0

out = `.`

Result: success

Vollgeschachtelt

p_in = `(((((((((.)))))))))`

i = 6

out = `((.))`

Result: success

Out of bounds

p_in = `(((...)))`

i = 20

out = `nullptr`

Result: success

Nur punkte

p_in = `.......`

i = 2

out = `.......`

Result: success

Beispiel 4

Lösungsansatz

Wir definieren eine Vergleichsfunktion. Diese vergleicht zwei Strings von links nach rechts basierend auf `char` *ASCII* Werten.

Diese Funktion werden wir benutzen, um einen Bubble Sort Algorithmus zu implementieren. Der Bubble Sort Algorithmus wird benutzt, weil er sehr einfach auf einen bestimmten Wertebereich einschrenkbar ist.

Dabei werden Elemente solange vertauscht, sofern sie nicht in der richtigen Reihenfolge sind, bis sie in einem sortierten Zustand vorliegen. Das ist nach spätestens n^2 Vergleichen der Fall.

Testfälle

Normaler Fall

vec = `[banana, apple, cherry, orange, date]`

start = 0 end = 4

out = `[apple, banana, cherry, date, orange]`

Result: success

Unterbereich

vec = `[zebra, yellow, white, violet, blue]`

start = 2 end = 4

out = `[zebra, yellow, blue, violet, white]`

Result: success

Ungültiger Bereich

vec = `[zebra, yellow, white, violet, blue]`

start = 3 end = 1

out = `Error: range is invalid`

Result: success

Leerer Vektor

vec = `[]`

start = 0 end = 0

out = `Error: vector is empty`

Result: success

Vektor mit nullptr

vec = `[apple, nullptr, cherry]`

start = 0 end = 2

out = `Error: null_ptr at index 1`

Result: success