

Erzeugung von Permutationen mit „Next Permutation“

1. Was ist eine Permutation

Eine n -stellige *Permutation* ist eine bestimmte Anordnung von n Elementen.

Formal: Sei $M = \{x_1, x_2, \dots, x_n\}$ eine Menge mit n Elementen. Dann ist eine n -stellige Permutation eine bijektive (eindeutige) Abbildung

$$\pi: M \rightarrow M$$

die jedem Element der Menge ein Element der gleichen Menge zuordnet.

Betrachten wir zum Beispiel die drei Elemente $(1, 2, 3)$. Eine mögliche Permutation ist $(2, 3, 1)$ und eine andere ist $(3, 1, 2)$.

2. Darstellung von Permutationen

- In der *Zweizeilenform* stehen in der oberen Zeile die Elemente $1, 2, \dots, n$ in aufsteigender Reihenfolge während darunter die Elemente stehen, auf die sie abgebildet werden.

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 2 & 6 & 1 & 4 & 3 \end{pmatrix}$$

- In der *Tupelschreibweise* denken wir uns die obere (sortierte) Zeile und notieren nur die untere Zeile der Zweizeilenform.

$$\pi = (5 \ 2 \ 6 \ 1 \ 4 \ 3)$$

- In der *Zyklenschreibweise* fassen wir diejenigen Elemente zusammen, die einen Zyklus bilden.

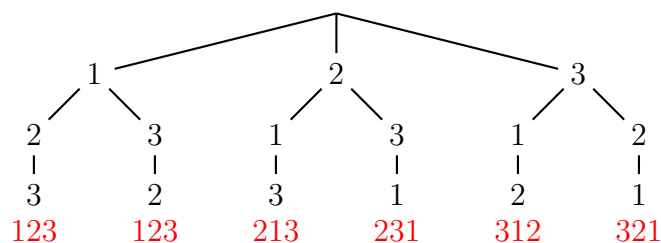
$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 2 & 6 & 1 & 4 & 3 \end{pmatrix} \Rightarrow (1 \ 5 \ 4) (2) (3 \ 6)$$

3. Anwendungen

- Brute-Force-Algorithmen, bei denen alle möglichen Anordnungen ausprobiert werden
- Suchen nach einer optimalen Reihenfolge, beispielsweise bei Planungs- oder Optimierungsproblemen

4. Anzahl Permutationen

Wir bestimmen diese Anzahl mit einem Permutationsbaum z. B. für $n = 3$ Elemente:



Liest man die Zahlen entlang der Pfade von der Wurzel (oben) bis zu den Blättern (ganz unten), so erhält man alle Permutationen aus drei Elementen.

Anzahl Permutationen von n Elementen: $n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1 = n!$

n	0	1	2	3	4	5	6	...
$n!$	1	1	2	6	24	120	720	...

$n!$ [lies: „ n Fakultät“] *Problem: $n!$ wächst extrem schnell.*

5. Lexikographische Ordnung

Eine Permutation α ist lexikographisch kleiner als eine Permutation β (kurz: $\alpha < \beta$), wenn das Zeichen $\alpha[i]$ mit dem niedrigsten Index i , in dem sich die beiden Permutationen unterscheiden, kleiner ist als das entsprechende Zeichen $\beta[i]$.

Beispiele:

$$(a) (3, 4, 5, \textcolor{red}{1}, 2) < (3, 4, 5, \textcolor{red}{2}, 1)$$

$$(b) (\textcolor{red}{3}, 1, 2, 3, 4) > (\textcolor{red}{2}, 5, 4, 3, 1)$$

6. Der Next Permutation-Algorithmus

Ziel: Bestimme zu einer Permutation die lexikographische nächste.

Input: Wähle eine Startpermutation $L = (1, 3, 5, 4, 2)$

- (1) Suche von rechts den grössten Index i mit $L[i] < L[i + 1]$. Ist das nicht möglich, ist L die letzte Permutation.

$$(1, \textcolor{red}{3}, 5, 4, 2) \text{ für } i = 1 \text{ gilt } \textcolor{red}{L[i]} < L[i + 1] \text{ bzw. } \textcolor{red}{3} < 5$$

- (2) Suche von rechts den grössten Index j mit $L[i] < L[j]$.

$$(1, \textcolor{red}{3}, 5, \textcolor{blue}{4}, 2) \text{ für } i = 1 \text{ und } j = 3 \text{ gilt } \textcolor{red}{L[i]} < \textcolor{blue}{L[j]} \text{ bzw. } \textcolor{red}{3} < \textcolor{blue}{4}$$

- (3) Tausche die Elemente an den Positionen i und j .

$$(1, \textcolor{red}{3}, 5, \textcolor{blue}{4}, 2) \Rightarrow (1, \textcolor{blue}{4}, 5, \textcolor{red}{3}, 2)$$

- (4) Kehre die Reihenfolge der Elemente $L[i + 1], \dots, L[n - 1]$ um.

$$(1, 4, \textcolor{red}{5}, \textcolor{red}{3}, 2) \Rightarrow (1, 4, \textcolor{red}{2}, \textcolor{red}{3}, 5)$$

$(1, 4, 2, 3, 5)$ ist die gesuchte lexikographisch nächste Permutation.

Beispiel: Erzeugung aller Permutationen der Länge 3

Beachte dass die Indizes i und j jeweils bei 0 beginnen.

Permutation	Schritt 1	Schritt 2	Schritt 3	Schritt 4
(1, 2, 3)	$i = 1$	$j = 2$	(1, 3 , 2)	(1, 3, 2)
(1, 3, 2)	$i = 0$	$j = 2$	(2 , 3 , 1)	(2, 1 , 3)
(2, 1, 3)	$i = 1$	$j = 2$	(2, 3 , 1)	(2, 3 , 1)
(2, 3, 1)	$i = 0$	$j = 1$	(3 , 2 , 1)	(3 , 1 , 2)
(3, 1, 2)	$i = 1$	$j = 2$	(3, 1 , 2)	(3, 2 , 1)
(3, 2, 1)	–			

Das Verfahren geht auf den Indischen Mathematiker Nārāyaṇa Paṇḍita (1340–1400) zurück und wurde seither oft wiederentdeckt.

7. Implementierung (Pseudocode)

Wir teilen die Aufgabe, alle Permutationen der Länge n zu erzeugen, in drei Teile auf:

1. die Funktion `next_permutation(L)`,
2. die Hilfsfunktion `reverse_slice(L, i, j)`,
3. die Verarbeitung aller Permutationen

7.1 Die Funktion next_permutation(L)

Die Funktion verändert die Liste L inplace und gibt **True** zurück, wenn eine nächste Permutation gefunden wurde. Existiert keine nächste Permutation, gibt sie **False** zurück.

```
1 FUNCTION next_permutation(L):
2     n <- length(L) // Schritt 1
3     i <- n - 2
4     WHILE i >= 0 AND L[i] >= L[i + 1]:
5         i <- i - 1
6     IF i < 0:
7         RETURN False
8
9     j <- n - 1 // Schritt 2
10    WHILE NOT (L[i] < L[j]):
11        j <- j - 1
12
13    SWAP L[i], L[j] // Schritt 3
14
15    reverse_slice(L, i + 1, n) // Schritt 4
16    RETURN True
```

7.2 Die Funktion reverse_slice(L, i, j)

Die Funktion `reverse_slice(L, i, j)` kehrt in der Liste L die Reihenfolge der Elemente mit den Indizes $i, i + 1, \dots, j - 1$ um.

```
1 FUNCTION reverse_slice(L, i, j)
2     left <- i
3     right <- j-1
4     WHILE left < right:
5         SWAP L[left], L[right]
6     left <- left + 1
7     right <- right - 1
```

7.3 Die Verarbeitung von Permutationen

Falls nötig, wird die Liste mit den Elementen aufsteigend sortiert. Anschliessend wird im Rumpf einer fustgesteuerten Schleife die Liste verarbeitet, dann mit `next_permutation(L)` die nächste Permutation bestimmt. Sobald der Rückgabewert dieser Funktion **false** zurückgibt, wird die Schleife beendet.

```
1 SORT(L)
2
3 WHILE true
4     PROCESS(L)
5     IF next_permutation(L) = false THEN
6         BREAK
```