Permutation angewandte Mathematik Fr. Dr. Logofatu

Benedikt Zönnchen, Erik Wenzel

1. Juli 2011

Inhaltsverzeichnis

Permutation ohne Wiederholung

Definition

Permutation von unterschiedlichen Elementen ohne Wiederholung, also eine bijektive Abbildung $A_n \to A_n$ einer n-elementigen Menge A_n auf sich selbst.

Permutation ohne Wiederholung

Definition

Permutation von unterschiedlichen Elementen ohne Wiederholung, also eine bijektive Abbildung $A_n \to A_n$ einer n-elementigen Menge A_n auf sich selbst.

Formel

Es gilt: Für n Elemente gibt es $n!$ Möglichkeiten.

Permutation ohne Wiederholung

Definition

Permutation von unterschiedlichen Elementen ohne Wiederholung, also eine bijektive Abbildung $A_n \to A_n$ einer n-elementigen Menge A_n auf sich selbst.

Formel

Es gilt: Für n Elemente gibt es $n!$ Möglichkeiten.

Beweis

$n = 1$: wahr, da ich ein Element nicht permutieren kann.
$n \to n + 1$: kommt ein Element hinzu so kann man alle neuen Permutationen erreichen indem man das Element durch alle neuen $n + 1$ Positionen aller vorherigen Permutationen wandern lässt.
$\Rightarrow n! * (n + 1) = (n + 1)!$

Permutation mit Wiederholung

Definition

Gegeben sei eine Menge A_n und wir teilen deren Elemente in Gruppen ein. Gesucht ist nun die Anzahl der Anordnungen, für die nur diese Eigenschaft relevant sind. Die Elemente einer Gruppe sind austauschbar.

Permutation mit Wiederholung

Definition

Gegeben sei eine Menge A_n und wir teilen deren Elemente in Gruppen ein. Gesucht ist nun die Anzahl der Anordnungen, für die nur diese Eigenschaft relevant sind. Die Elemente einer Gruppe sind austauschbar.

Formel

Es gilt:

$$\frac{n!}{n_1!n_2!...n_k!}$$

n: Anzahl der Elemente
k: Anzahl der Gruppen
n_k: Anzahl der Elemente der Gruppe.

Permutation mit Wiederholung

Beispiel

Nehmen wir an wir haben 8 Kugeln. 3 weiße, 4 schwarze und 1 blaue. So ist die Anzahl der Möglichkeiten an Anordnungen $=$

$$\frac{8!}{3!4!1!}$$

Begriffe

Rang

Wenn wir alle lexikographisch geordneten Permutationen von 0 bis $n-1$ durchnummerieren so ist diese Nummer der Rang der Permutation.

Begriffe

Rang

Wenn wir alle lexikographisch geordneten Permutationen von 0 bis $n-1$ durchnummerieren so ist diese Nummer der Rang der Permutation.

π

π ist unsere Permutation und $\pi(x)$ ist das Element an der Stelle x in der Permutation.

ID Codes (146)

Aufgabe

Wir bekommen als Eingabe eine Zeichenfolge s wobei $|s| \leq 10$ ist.
Unsere Aufgabe ist es die Nachfolgepermutation zu finden.

ID Codes (146)

Aufgabe

Wir bekommen als Eingabe eine Zeichenfolge s wobei $|s| \leq 10$ ist.
Unsere Aufgabe ist es die Nachfolgepermutation zu finden.

Beispiel

Eingabe = abc $\Rightarrow$ Ausgabe = acb
Eingabe = accba $\Rightarrow$ Ausgabe = baacc

ID Codes (146)

Lösungsalgorithmus

1. Suche von rechts das erste Element p_k, das $p_k < p_{k+1}$ erfüllt

ID Codes (146)

Lösungsalgorithmus

1. Suche von rechts das erste Element p_k, das $p_k < p_{k+1}$ erfüllt
2. Such erneut von rechts das erste Element p_t, das $p_t > p_k$ erfüllt

ID Codes (146)

Lösungsalgorithmus

1. Suche von rechts das erste Element p_k, das $p_k < p_{k+1}$ erfüllt
2. Such erneut von rechts das erste Element p_t, das $p_t > p_k$ erfüllt
3. Vertausche p_k und p_t

ID Codes (146)

Lösungsalgorithmus

1. Suche von rechts das erste Element p_k, das $p_k < p_{k+1}$ erfüllt
2. Such erneut von rechts das erste Element p_t, das $p_t > p_k$ erfüllt
3. Vertausche p_k und p_t
4. Kehre die Sequenz $p_{k+1}p_{k+2}...p_n$ um

ID Codes Quellcode

```
1   private List<T> list;
2   ...
3   // suche von rechts p_k > p_{k+1}
4   private int getPivot(){
5           int i = list.size() - 1;
6           while (i > 0 && list.get(i-1).compareTo(list.get(i)) > -1){
7               i--;
8           }
9           return --i;
10  }
11  // vertausche p_i mit p_j
12  public void swap(int i, int j){
13          T tmp = list.get(i);
14          list.set(i, list.get(j));
15          list.set(j, tmp);
16  }
```

ID Codes Quellcode

```
1   public List<T> nextPermutation(){
2           if(list.size() <= 1){
3               return null;
4            }
5            // suche von rechts p_k > p_{k+1}
6           int pivot = getPivot();
7           if(pivot>=0){
8                // suche erneut von rechts p_t > p_k
9               int i = list.size() - 1;
10              while (i > 0 && list.get(pivot).compareTo(list.get(i)) > -1){
11                   i--;
12              }
13              // vertausche p_k mit p_t
14              swap(pivot, i);
15              // drehe die Sequenz ab p_{k+1} bis zum ende um
16              Collections.reverse(list.subList(pivot + 1, list.size()));
17          }
18          else{
19              return null;
20          }
21          return list;
22  }
```

Ranking

Aufgabe

Wir bekommen n Zahlen von 1 bis n (also eine Permutation) und müssen entscheiden welchen Rang diese Permutation hat.

Ranking

Aufgabe

Wir bekommen n Zahlen von 1 bis n (also eine Permutation) und müssen entscheiden welchen Rang diese Permutation hat.

Beispiel

Eingabe $= 1\ 2\ 3\ 4 \Rightarrow$ Ausgabe $= 0$
Eingabe $= 1\ 2\ 4\ 3 \Rightarrow$ Ausgabe $= 1$

Ranking

Lösungsansatz

Rang				
0:	1	2	...	n
1:	1	2	...	
...:	1	...	...	
$(n-1)! - 1$:	1	...	...	
$(n-1)!$:	2	...	...	
...:	2	...	...	
$2(n-1)!$:	3	...	...	

Ranking

Lösungsansatz

Rang

0:	1	2	...	n
1:	1	2	...	
...:	1	...	...	
$(n-1)! - 1$:	1	...	...	
$(n-1)!$:	2	...	...	
...:	2	...	...	
$2(n-1)!$:	3	...	...	

$$\Rightarrow (\pi(1) - 1)(n-1)! \leq rank(\pi) \leq \pi(1)(n-1)! - 1$$

Ranking

Beispiel

$n = 4$. Wir suchen den Rang der Permutation 1 3 4 2. Durch die Abschätzung wissen wir schon wo der Rang in etwa liegen wird.

Ranking

Beispiel

$n = 4$. Wir suchen den Rang der Permutation 1 3 4 2. Durch die Abschätzung wissen wir schon wo der Rang in etwa liegen wird.

$$(1-1)(4-1)! \geq rank(\pi) \geq 1(4-1)! - 1$$

$$\Rightarrow 0 \geq rank(\pi) \geq 5$$

Ranking

Beispiel

$n = 4$. Wir suchen den Rang der Permutation 1 3 4 2. Durch die Abschätzung wissen wir schon wo der Rang in etwa liegen wird.

$$(1-1)(4-1)! \geq rank(\pi) \geq 1(4-1)! - 1$$

$$\Rightarrow 0 \geq rank(\pi) \geq 5$$

Rang				
0:	1	2	3	4
1:	1	2	4	3
2:	1	3	2	4
3:	1	3	4	2
4:	1	4	2	3
5:	1	4	3	2

Ranking

Lösung

Daraus folgt dann die Formel:

$$rank(\pi, n) = (\pi(1) - 1)(n-1)! + rank(\pi', n-1)$$

Ranking

Lösung

Daraus folgt dann die Formel:

$$rank(\pi, n) = (\pi(1) - 1)(n-1)! + rank(\pi', n-1)$$

Rang					
0:	1	2-1= 1	3-1= 2	4-1= 3	
1:	1	2-1= 1	4-1= 3	3-1= 2	
2:	1	3-1= 2	2-1= 1	4-1= 3	$\Rightarrow \pi'$
3:	1	3-1= 2	4-1= 3	2-1= 1	
4:	1	4-1= 3	2-1= 1	3-1= 2	
5:	1	4-1= 3	3-1= 2	2-1= 1	

Ranking

Lösung

Daraus folgt dann die Formel:

$$rank(\pi, n) = (\pi(1) - 1)(n-1)! + rank(\pi', n-1)$$

Rang					
0:	1	2-1= 1	3-1= 2	4-1= 3	
1:	1	2-1= 1	4-1= 3	3-1= 2	
2:	1	3-1= 2	2-1= 1	4-1= 3	$\Rightarrow \pi'$
3:	1	3-1= 2	4-1= 3	2-1= 1	
4:	1	4-1= 3	2-1= 1	3-1= 2	
5:	1	4-1= 3	3-1= 2	2-1= 1	

$$\Rightarrow rank(\pi, n) = \sum_{i=1}^{n} n_i * (n-i)!$$

Ranking Quellcode

```
1   private List<T> list;
2   ...
3   public long rank(){
4           int n = list.size();
5           int ni;
6           long rank = 0;
7           long fact = 1;
8
9           for(int i=n-2; i >= 0; i--){
10              fact *= n-i-1;
11              ni = 0;
12              for(int j = i+1; j < n; j++) {
13                  if(list.get(j).compareTo(list.get(i)) == -1) {
14                      ni++;
15                  }
16              }
17              rank += ni*fact;
18          }
19          return rank;
20  }
```

$$\sum_{i=1}^{n} n_i(n-i)!$$

Unranking (941)

Aufgabe

Wir bekommen n und *rank*. n ist die Anzahl der Elemente und *rank* ist der Rang. Wir sollen daraus die Permutation errechnen. (Eigentlich ging es in der Aufgabe um Buchstaben aber dies läuft aufs gleich hinaus)

Unranking (941)

Aufgabe

Wir bekommen n und *rank*. n ist die Anzahl der Elemente und *rank* ist der Rang. Wir sollen daraus die Permutation errechnen. (Eigentlich ging es in der Aufgabe um Buchstaben aber dies läuft aufs gleich hinaus)

Beispiel

Eingabe $= 4\ 0 \Rightarrow$ Ausgabe $= 1\ 2\ 3\ 4$
Eingabe $= 3\ 1 \Rightarrow$ Ausgabe $= 1\ 3\ 2$

Unranking (941)

Lösungsansatz

Wir wissen, dass die $\pi(1)$, $(n-1)!$-mal 1 ist bevor $\pi(1) = 2$ wird. Wir können also die erste Stelle berechnen und dann den Rang dementsprechend verringern und mit dem Rest weiter rechnen. Dies kann wieder rekursiv durchgeführt werden.

Unranking (941)

Beispiel

Angenommen $n = 4$ und Rang $= 7$.

$3! = 6 < 7$

$\Rightarrow$ erste Zahl $= 1 + \frac{7}{6} = 2$, als Restrang bleibt $(7 \mod 6) = 1$

Rang				
6:	2	1	3	4
7:	2	1	4	3
8:	2	3	1	4
9:	2	3	4	1
10:	2	4	1	3
11:	2	4	3	1

Unranking (941)

Beispiel

Angenommen $n = 4$ und Rang $= 7$.

$3! = 6 < 7$

$\Rightarrow$ erste Zahl $= 1 + \frac{7}{6} = 2$, als Restrang bleibt $(7 \mod 6) = 1$

Rang						Restrang			
6:	2	1	3	4		0:	1	3	4
7:	2	1	4	3		1:	1	4	3
8:	2	3	1	4	$\Rightarrow$	2:	3	1	4
9:	2	3	4	1		3:	3	4	1
10:	2	4	1	3		4:	4	2	3
11:	2	4	3	1		5:	4	3	1

Unranking (941)

Lösung

(grob)

1. i = 0 bis n

Unranking (941)

Lösung

(grob)

1. i = 0 bis n
2. Ziffer = Rang / (n-i-1)! und den Rang = Rang mod (n-i-1)!

Unranking (941)

Lösung

(grob)

1. i = 0 bis n
2. Ziffer = Rang / (n-i-1)! und den Rang = Rang mod (n-i-1)!
3. Überspringe alle schon gesetzten Ziffern

Unranking (941)

Lösung

(grob)

1. i = 0 bis n
2. Ziffer = Rang / (n-i-1)! und den Rang = Rang mod (n-i-1)!
3. Überspringe alle schon gesetzten Ziffern
4. Merke dir die Ziffer als bereits gesetzt

Unranking (941)

Lösung

(grob)

1. i = 0 bis n
2. Ziffer = Rang / (n-i-1)! und den Rang = Rang mod (n-i-1)!
3. Überspringe alle schon gesetzten Ziffern
4. Merke dir die Ziffer als bereits gesetzt
5. Gehe zu 1.

Unranking Quellcode

```
1   private List<T> list;
2   ...
3   public void unrank(long p) {
4       long rank = p;
5       int n = list.size();
6       List<T> copy = new ArrayList<T>(n);          // Hilfsliste zum merken der belegten Positionen
7       List<T> target = new ArrayList<T>(n);        // Ergebnisliste
8       long[] fak = getFakArray(n);                 // generiere ein Hilfsarray mit den notwendigen n!
9
10      for(int i = 1; i <= n; i++)                  // Setzte alle Positionen auf unbesetzt
11      {
12          copy.add(null);
13      }
14
15      for (int i = 0; i < n; i++)                  // von 0 bis n-1
16      {
17          int position = (int) (rank / fak[n-i-1]); // berechne die Ziffer
18          rank = rank % fak[n-i-1];                // generiere Restrangrang
19
20          int freePosition = 0;
21          while (position >= 0 ){                  // suche die richtige Ziffer
22              if (copy.get(freePosition++) == null){
23                  position--;
24              }
25          }
26          copy.set(freePosition-1, list.get(freePosition-1)); // setzte die Position auf besetzt
27          target.add(list.get(freePosition-1));    // setzte richtige Ziffer in Permutation
28       }
29       list = target;}                             // setzte fertig generierte Permutation
```

Quellen

- Algorithmen und Problemlösungen mit C++ von Doina Logofatu
- Mathematik für Informatiker von Peter Hartmann
- UVa Online Judge