KAPITEL 5: SORTIEREN

5.0 Vorspann: Sortieren von Übungszetteln

5.1 Sortieren und Permutationen

Gegeben: Eine Menge von Objekten $x_1, \dots, x_n$,
eine Größenrelation „$<$", die
je zwei Objekte ordnet
(→ Totalordnung!)

Gesucht: Eine Sortierung der Objekte nach
Reihenfolge.

Bemerkungen:

(1) Wir schreiben der Einfachheit halber

$$x_i < x_j$$

bzw $x_j < x_i$

(und gehen wie gesagt davon aus,
das eines davon gilt)

(2) Statt die Objekte in der vorgegebenen
Reihenfolge hinzuschreiben, reicht es auch,
die Permutation zu kennen, die
die Anordnung beschreibt:

Bauer (2) Meyer (5/4) Konrad (3) Anton (1) Maier (5)

↳ Anton (1) Bauer (2) Konrad (3) Maier (4) Meyer (5)

Also hat man Zugriff in sortierter Form durch Codierung

4 1 3 2 5

(Verwirrend? Konzentrationssache!)

Schreibweise für Permutationen:

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

Bedeutung: Interpretation als Pointer

Größtes Objekt an Stelle 4
Zweitgrößtes Objekt an Stelle 1
Drittgrößtes Objekt an Stelle 3
Viertgrößtes Objekt an Stelle 2
Fünftgrößtes Objekt an Stelle 5

Interpretation als Transportanleitung; d.h.

Bringe Objekt aus 1 nach 4
aus 2 nach 1
aus 3 nach 3
aus 4 nach 2
aus 5 nach 5 .

Hintereinanderausführung:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \end{pmatrix} \quad \text{gefolgt von} \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

also:

Objekt aus 1 nach 4 nach 2
aus 2 nach 1 nach 5
aus 3 nach 3 nach 3
aus 4 nach 2 nach 4
aus 5 nach 5 nach 1

$$\rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 4 & 1 \end{pmatrix}$$

Sortieren als Transportvorgang:

Welche Permutation macht

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \end{pmatrix}$$ rückgängig?

1	nach	4	–	nach 1
2	nach	1	–	nach 2
3	nach	3	–	nach 3
4	nach	2	–	nach 4
5	nach	5	–	nach 5

Also $$\begin{pmatrix} 4 & 1 & 3 & 2 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

bzw. (sortiert!)

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

Also: - Zugriff in aktueller Anordnung in sortierter Form per Pointer:

Permutation der Objekte, also Anordnungspermutation

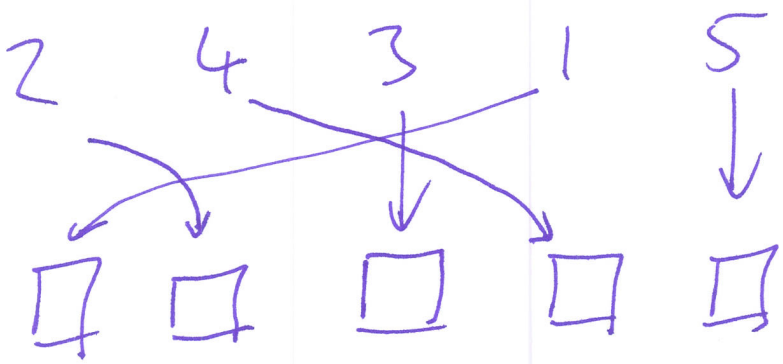
- Rückgängigmachen der Unordnung, um die Objekte explizit in sortierte Reihenfolge zu bringen:

Inverse Permutation der Objekte; d.h. Rücksortierungspermutation

5.2 Vorbetrachtungen zum Modell

Beim Sortieren gibt es wichtige Unterschiede!

(1) Kennt man explizit die Nummerierung der Objekte, dann geht es schnell:



also

(2) Kennt man keine explizite Nummerierung, sondern kann die Objekte nur paarweise vergleichen (Balkenwaage!), so wird es schwerer!

Beobachtungen:

(i) Schaut man sich bei Nummerierung ein Objekt an, reduziert sich die Zahl der Möglichkeiten für die Anordnungspermutation drastisch!

(Schaut man sich das erste Objekt an und identifiziert eine 2, so bleiben von $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ möglichen Anordnungen nur noch 24 Kandidaten übrig, danach noch 6, etc.!)

Das geht explizit in Linearzeit:

In Zelle Nr.	steht	gewünscht
1	$\pi(1)$	$\pi^{-1}(1)$
2	$\pi(2)$	$\pi^{-1}(2)$
3	$\pi(3)$	$\pi^{-1}(3)$
4	$\pi(4)$	$\pi^{-1}(4)$
5	$\pi(5)$	$\pi^{-1}(5)$

Also: Schreibe nach

$\pi(i)$ den Wert $\pi^{-1}(i)$

Äquivalent Schreibe nach

$\pi(\pi(i))$ den Wert i !

Also in Pseudocode:

```
FOR (i=1 TO n) DO {
    IP[P[i]] := i
}
```

(dabei P[.]: Permutation, IP[.]: inverse Permutation)

entspricht

```
FOR (i:=1 TO n) DO {
    P[P[i]] := i
```

(2) Kennt man keine explizite Nummerierung, sondern kann die Objekte nur paarweise vergleichen (Balkenwaage o.ä.!), so wird es schwerer!

Beobachtung:

(ii) Vergleicht man zwei Objekte (x_i und x_j) in der Anordnung, so teilt man die Menge der ~~Permutationen~~ möglichen Permutationen in zwei Mengen:

(>) Die Permutationen, bei denen x_i vor x_j steht;

(<) die Permutationen, bei denen x_j vor x_i steht.

Wie schwerwiegend ist das?

Wenn man Pech hat, ist hinterher die größere Menge übrig, also kann man auch bei clevererer Auswahl bestenfalls eine Reduktion um einen

Faktor 2 erreichen!

Satz 5.1

(1) Die Objekte $1, \dots, n$ kann man in Linearzeit sortieren, wenn man Arrays verwenden und in $O(1)$ direkt auf diese zugreifen darf.

(2) Für n Objekte $x_1, \dots, x_n$ benötigt man zum Sortieren mindestens $\Omega(n \log n)$, wenn man die Objekte nur paarweise vergleichen kann.

Beweis:

(1) Siehe oben!

(2) (a) Am Anfang hat man $n! = n \cdot (n-1) \cdot \dots \cdot 3 \cdot 2 \cdot 1$ viele mögliche Permutationen.

(b) Jeder Vergleich teilt die Menge der verbliebenen Permutationen in zwei Teilmengen

(c) Im schlechtesten Falle bleibt die größere Teilmenge übrig, na

(d) Man erreicht also bestenfalls eine Halbierung.

(e) Bis man eine eindeutige Permutation sicher identifiziert hat, braucht man also mindestens

$$\log_2(n!)$$

Vergleiche.

(f)
$$\log_2(n!) = \sum_{i=1}^{n} \log_2 n$$

$$\geq \sum_{i=\frac{n}{2}}^{n} \log_2 n$$

$$\geq \frac{n}{2}\left(\log_2 \frac{n}{2}\right)$$

$$= \frac{n}{2}\left(\log_2 n - 1\right)$$

$$\in \Omega(n \log n)$$

∎