

10. November

strings: Für Texte können wir den Typ `string` verwenden.

Wir brauchen dafür die Bibliothek `#include <string>`.

Es gilt $\text{string} \hat{=} \text{vector} < \text{char} >$

Warum also `string` und nicht einfach `vector`?

1. vergleichbar: `text1 == text2`
2. hintereinander hängen: `text1 += text2`
3. bequemes In-/Output: `cout << text;`

Matrizen: Matrizen sind eine Sammlung von Vektoren.

Form:

`std::vector < std::vector < int > > name (anz-zeil, std::vector < int > (anz-spalt, value))`

⇒ erzeugt eine Matrix der Grösse `anz-zeil` × `anz-spalt`, wobei alle den Eintrag `value` haben.

Die Einträge können wie folgt abgelesen werden

$$A_{ij} = \text{name.at}(i).\text{at}(j)$$

Anzahl Spalten: `name.at(0).size();`

Anzahl Zeilen: `name.size();`

Rekursion: In der Mathe: $f(n+1) = \begin{cases} 3f(n)+1 & \text{für } f(n) \text{ ungerade} \\ \frac{f(n)}{2} & \text{für } f(n) \text{ gerade} \end{cases}$

In C++ gleich! Wir können eine Funktion innerhalb dieser aufrufen. Bei vielen Problemen ist die Rekursion aber nicht gleich ersichtlich. Wie geht man vor?

1. Wir müssen das Problem für ein "simples" n lösen (bspw. $n=1$) **Base Case**
2. Sei $(n-1)$ als Lösung gegeben. In wiefern kann ich diese Lösung benutzen um n zu erhalten?
3. Zusammensetzen der rekursiven Funktion

Backtracking: Beim Backtracking - Algorithmus geht es um das "trial and error" - Prinzip. Der Algorithmus geht wie folgt vor:

1. Man setzt einen validen Wert und geht zum nächsten Feld und setzt wieder einen validen Wert ein usw.
2. Kann kein valides Wert gefunden werden, sucht man das Problem beim letzten Feld und setzt einen anderen validen Wert ein.

Anwendungen: Sudoku, Damenproblem, Wegsuche, Solitär