Übungsstunde

Woche 08

Adel Gavranović
adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Intro

Multidimensionale Vektoren

Rekursion

Outro

Links

- polybox zum Material für die Übungsstunden
- Mail an Assistenten

Follow-up aus vorheriger Übungsstunde

Informationen

- **alle** Basen (hexidez, octal, etc.) sind prürel
- für das Self-Assessment empfiehlt es sich, sich via *private Browsing* auf moodle anzumelden
- `j <= MAX_INT` müssen nicht zwingend in die PRE-condition
- falls Fragen zum Thema *Funktionen* oder *Vektoren* bestehen, so fragt unbedingt! (via Mail oder in der Stunde)
- benutzt `vec[i]` nicht, sondern `vec.at(i)`

Kommentare zu [code]expert

Allgemein

- Manchmal ist es echt einfach besser, eine Zwischenkopie zu machen und die Aufgabe einfach am nächsten Tag komplett neu anzugehen

E6:T1

- WOW! Sehr gute Leistung!
- returns können auch mitten in einer Funktion benutzt werden (spart manchmal ein paar Variablen)
- Viele *nested* `if()` lassen sich gut durch eine boolean expression vereinfachen
- unbedingt ML anschauen. Sehr vieles kann viel eleganter implementiert werden
- Wenn ihr wisst, dass ihr einen Wert irgendwo sehr oft verwendet werdet (insb. einen Wert, den ihr von einer Funktion erhaltet), dann speichert diesen in einer Variabel ab und nutzt ihn dann so

Kommentare zu [code]expert

E6:T2a

- ignoriert meine Kommentare zu fehlenden PRE und POSTs...
- Funktionen mit return type `bool` soll man am besten `is_adjective()` nennen. Also so etwas wie `is_inRange()` oder `is_valid()`

E6:T2b

- Schaut die ML unbedingt an
- PRE und POSTs können sich auch auf Sachen beziehen, die nicht Input der Funktion sind

E6:T3

- wenn etwas in sehr wenig Code erledigt werden kann und dabei lesbar/verständlich bleibt (mit oder ohne Kommentare), dann implementiert das so

Fragen zu [code]expert eurerseits?

Lernziele

- ☐ zweidimensionale Vektoren verstehen und in C++ implementieren können
- ☐ Rekursion verstehen können und Probleme mittels Rekursion angehen und lösen können
- ☐ Programme schreiben können, die mittels Rekursion *alle* möglichen Lösungen eines Problems aufzeigen

Was sind Multidimensionale Vektoren?

Multidimensionale Vektoren sind Matrizen

Aufgabe ”Matrix Transpose”

- öffnet ”Matrix Transpose” auf [code]expert

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^\top = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

- überlegt euch, wie ihr das Problem mit Stylus und Tablet angehen würdet
- Vereinfachung der Syntax:

```
using irow = std::vector<int>;
using imatrix = std::vector<irow>;
```

- Programmiert eine Lösung (optional in Gruppen)

Lösung zu ”Matrix Transpose”

```
imatrix transpose_matrix(const imatrix &matrix){
  unsigned int r, c;
  r = get_rows(matrix);          // number of rows
  c = get_cols(matrix);          // number of columns
  imatrix transposed_matrix;     // init' transp. matrix
  for(unsigned int col_i = 0; col_i < c; col_i++){
    irow row;                    // init' transp. row
    // entry-wise add transp. row to transp. matrix
    for(unsigned int row_i = 0; row_i < r; row_i++){
      row.push_back(matrix.at(row_i).at(col_i));
    }
    transposed_matrix.push_back(row);
  }
  return transposed_matrix;
}
```

Fragen/Unklarheiten?

Was ist Rekursion?

Rekursion

oft hilfreich für das Lösen von Problemen mit dem *divide and conquer-approach*

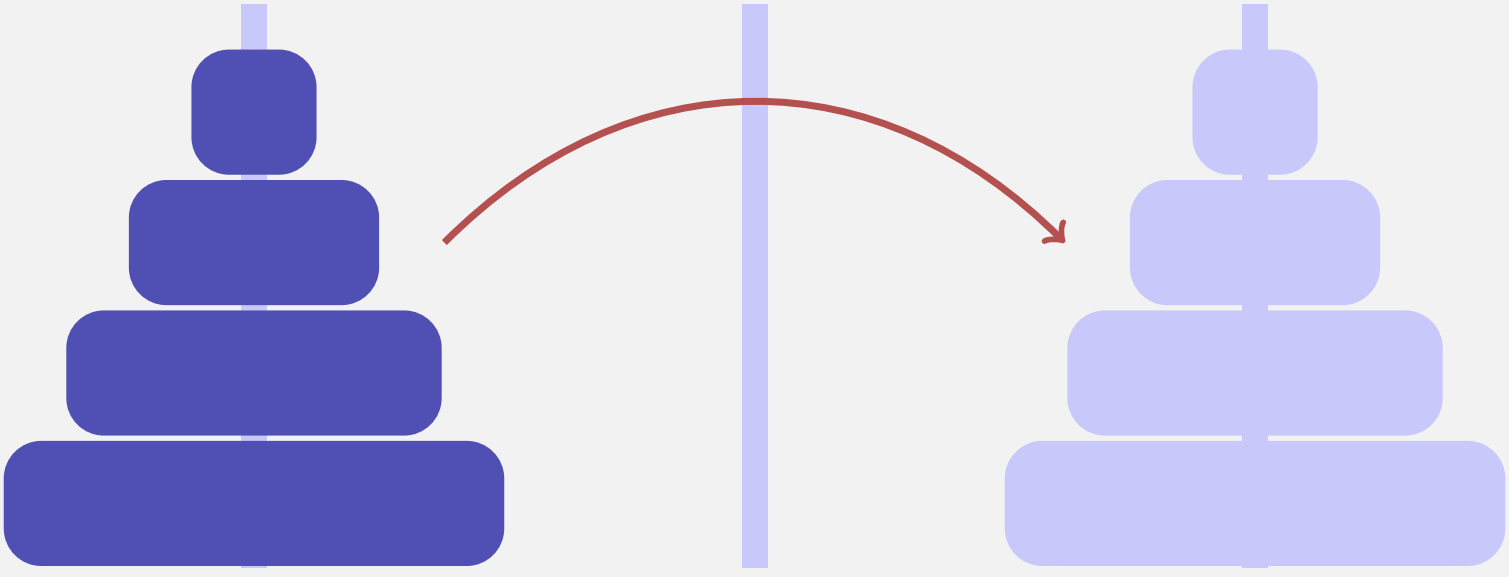
Wir wollen ein Problem für n lösen.

1. finde einen Weg, das Problem in kleinere Teilprobleme zu unterteilen: $k_0, k_1, \ldots, k_m \quad (\forall 0 \leq i \leq m : k_i < n)$
2. löse jedes k_i unabhängig voneinander (vielleicht durch weiteres Unterteilen)
3. setze die Lösung des Problems aus den Lösungen der Teilproblemen k_i zusammen

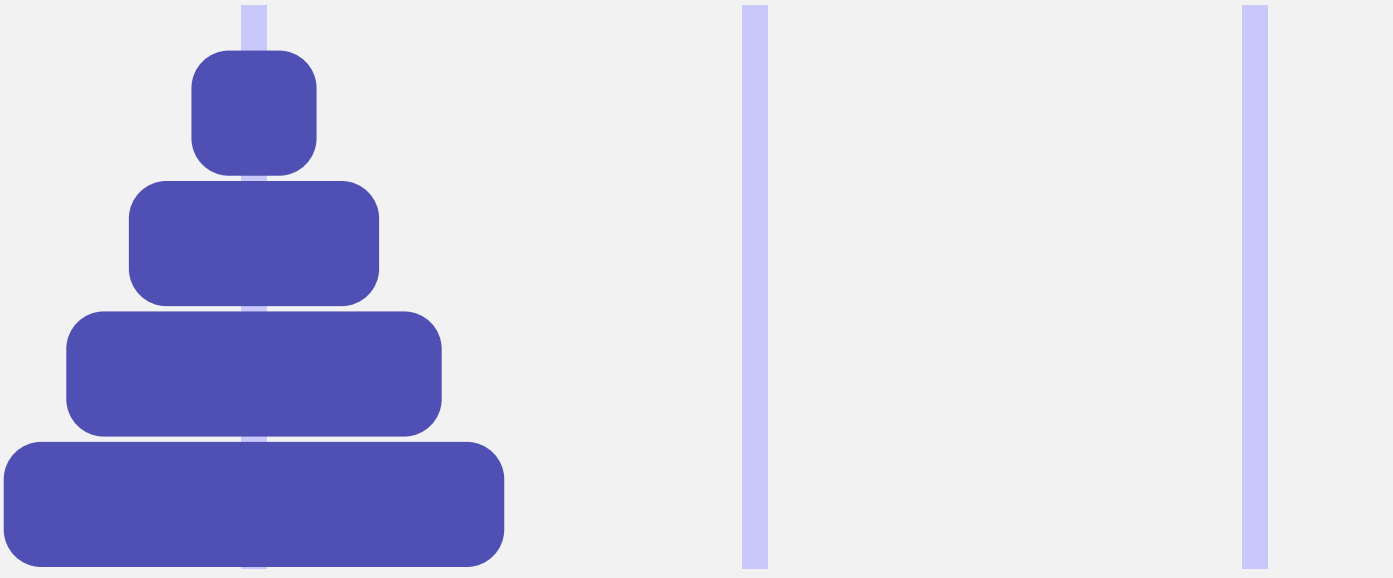
Experiment: Die Türme von Hanoi

Links Mitte Rechts

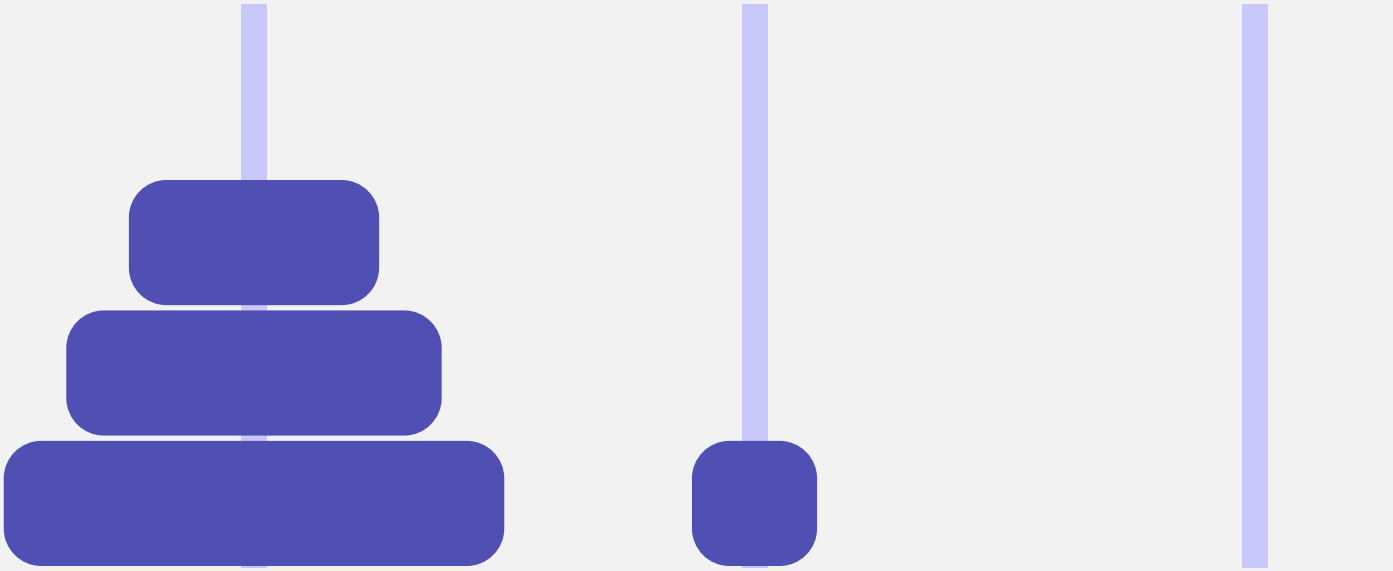
Experiment: Die Türme von Hanoi



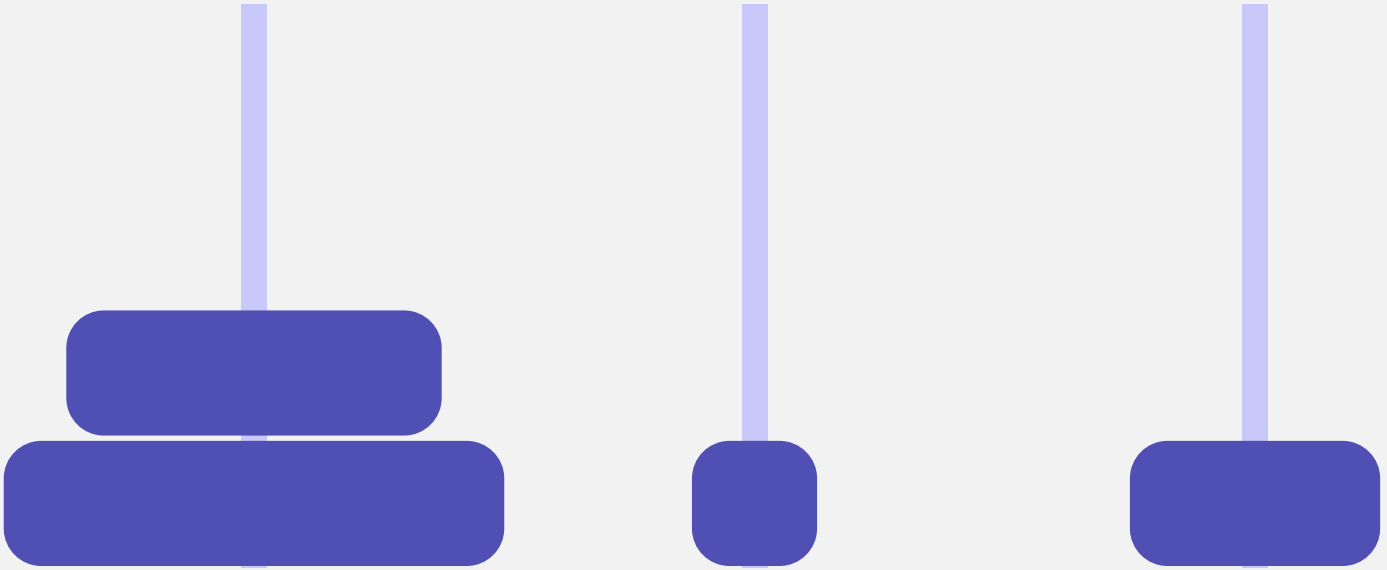
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!

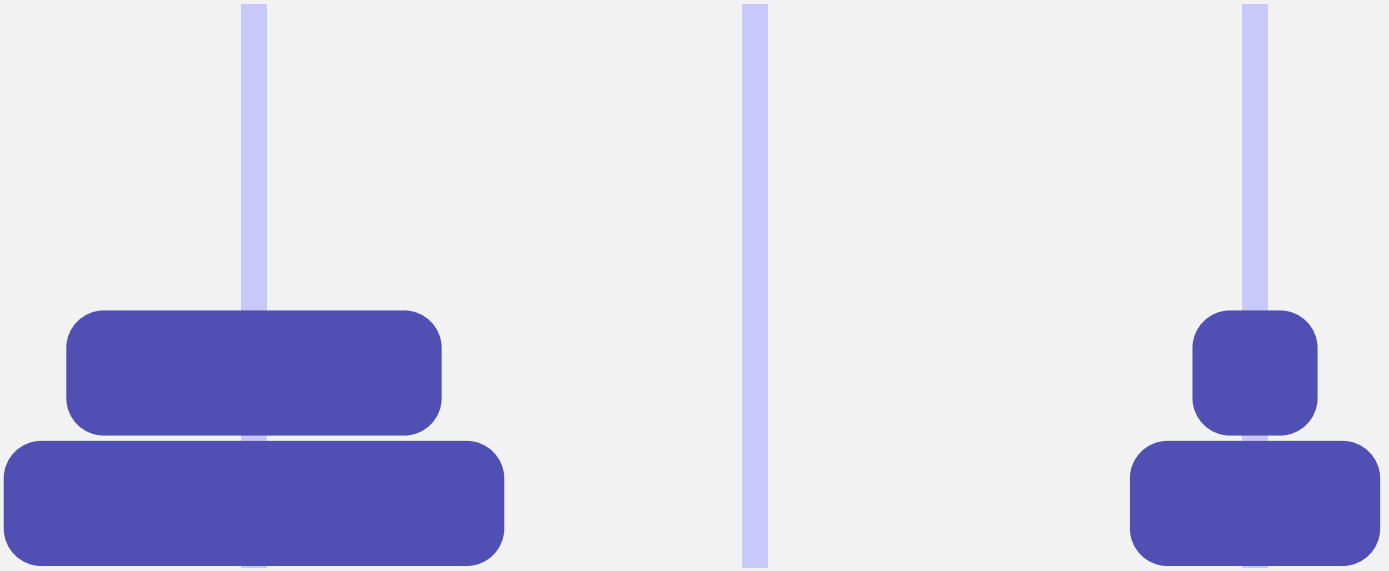


Die Türme von Hanoi - So gehts!



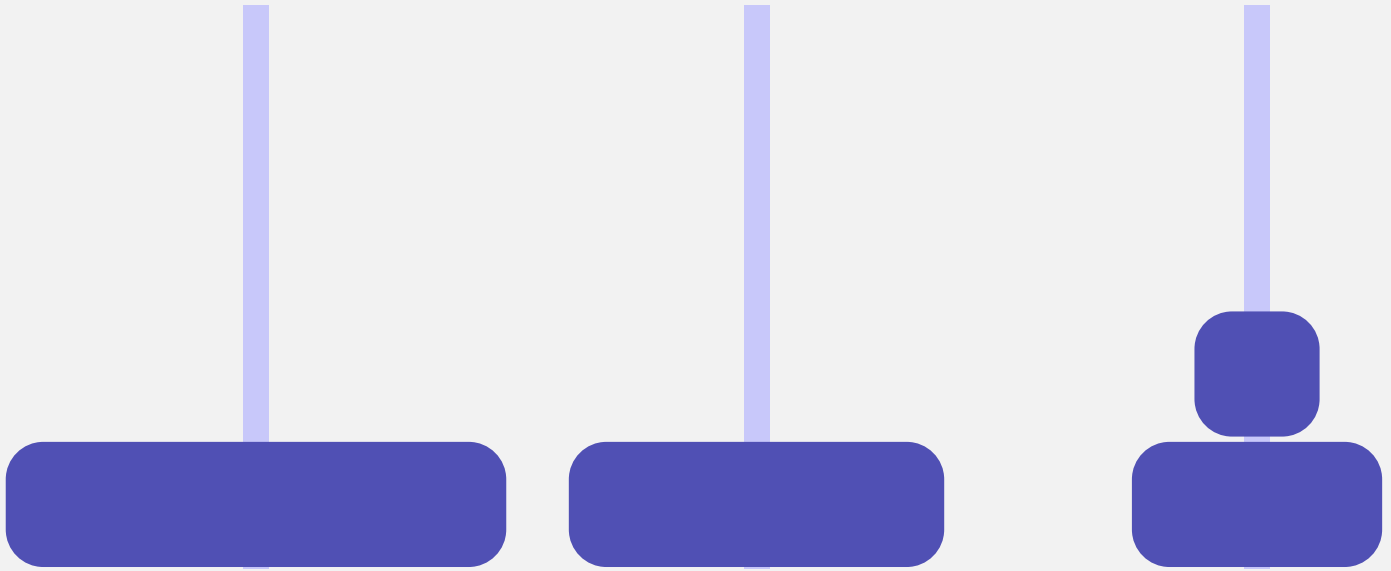
Links Mitte Rechts

Die Türme von Hanoi - So gehts!



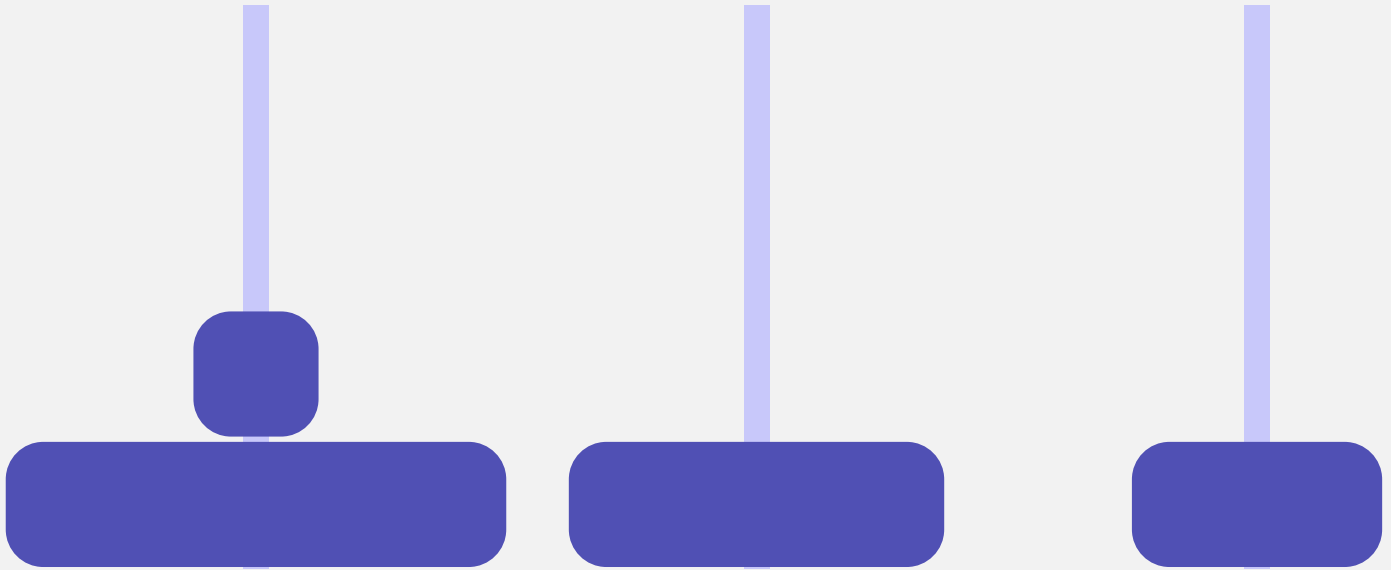
Links Mitte Rechts

Die Türme von Hanoi - So gehts!

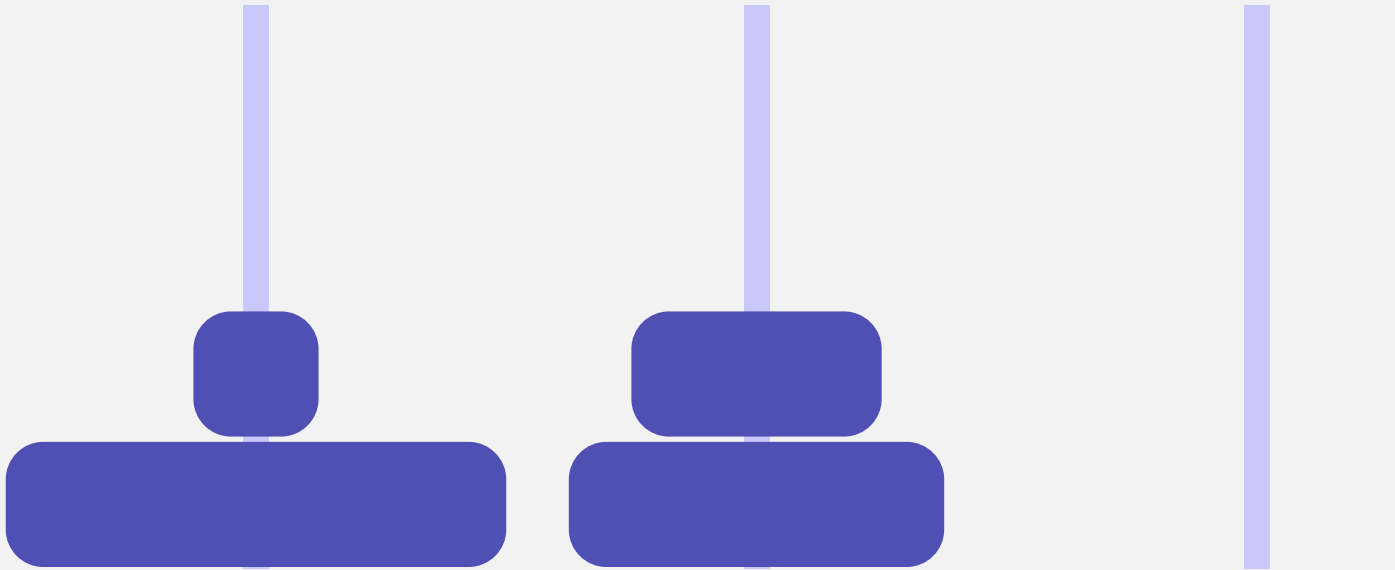


Links Mitte Rechts

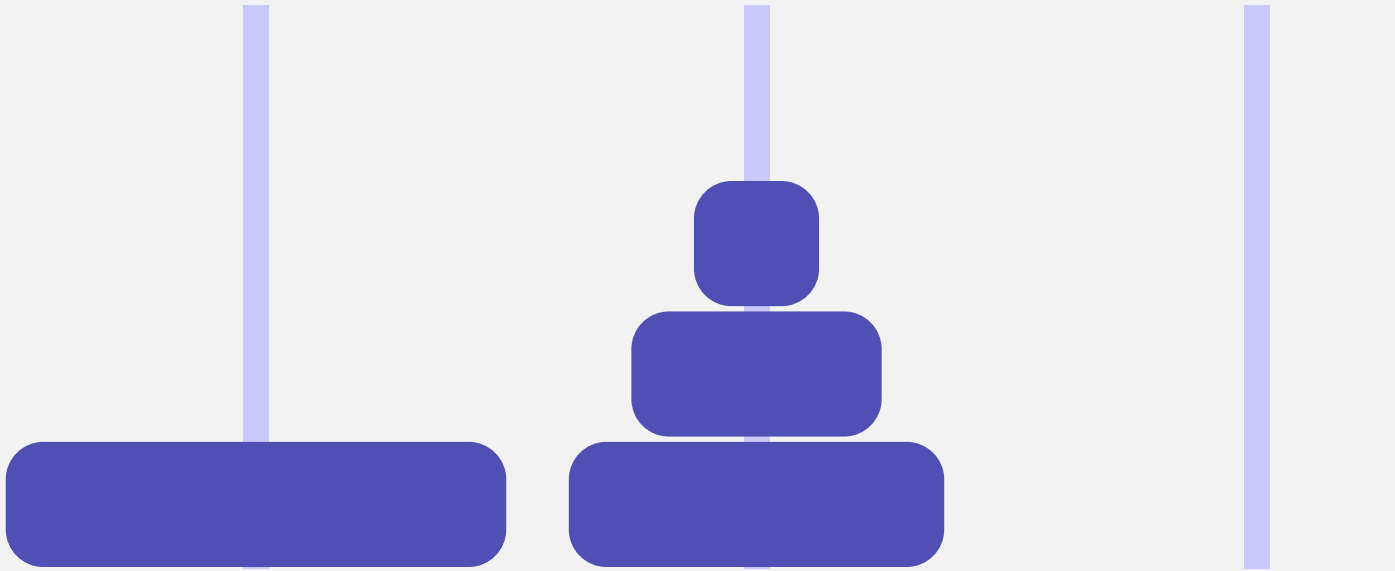
Die Türme von Hanoi - So gehts!



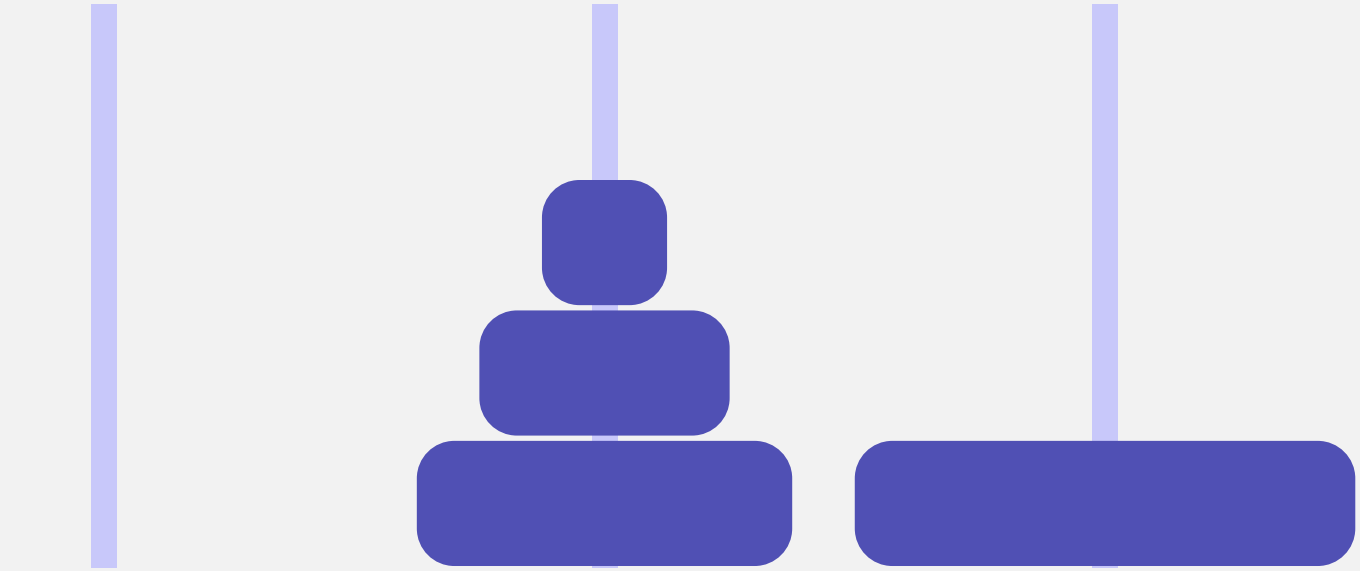
Die Türme von Hanoi - So gehts!



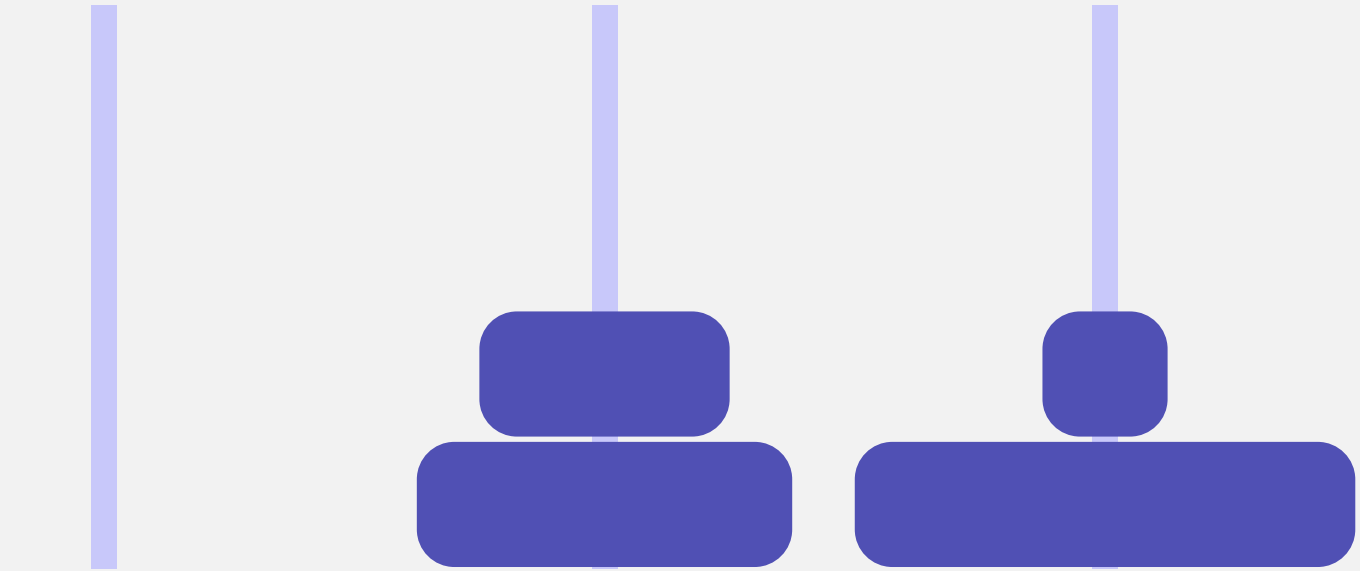
Die Türme von Hanoi - So gehts!



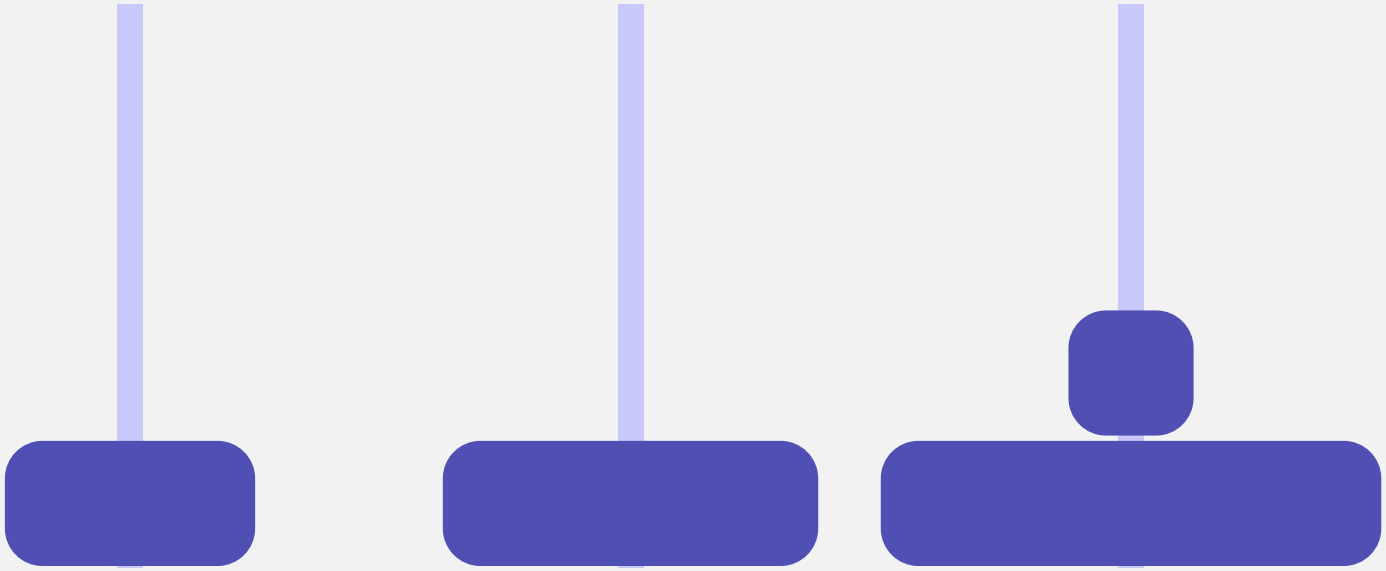
Die Türme von Hanoi - So gehts!



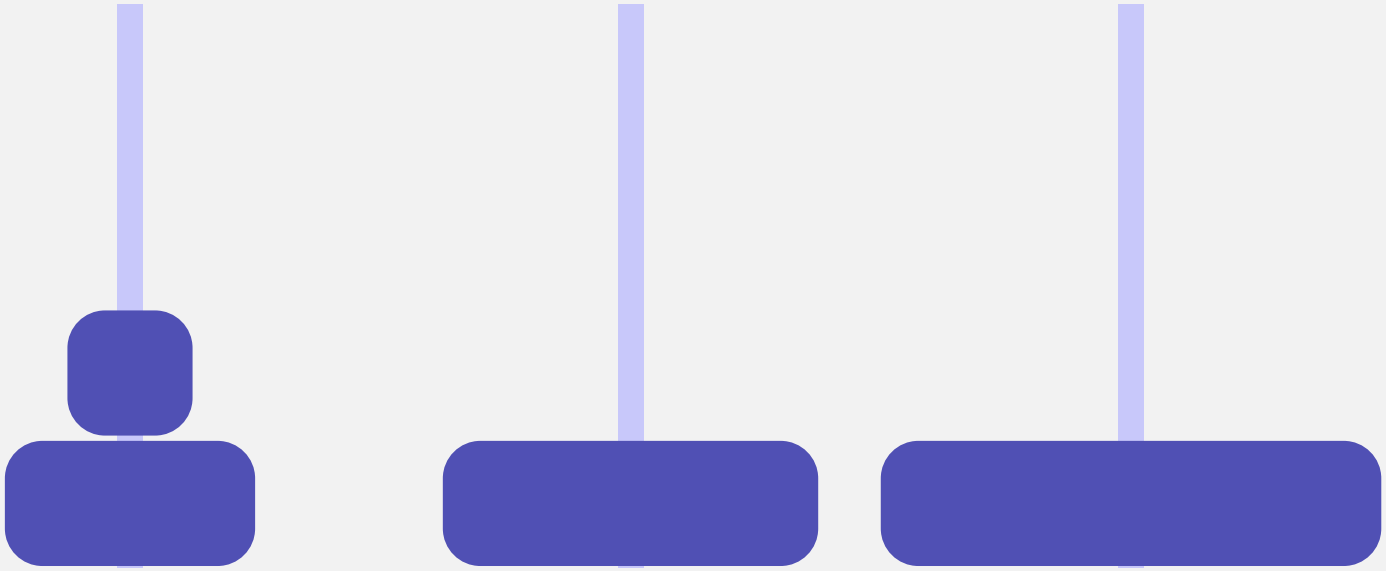
Die Türme von Hanoi - So gehts!



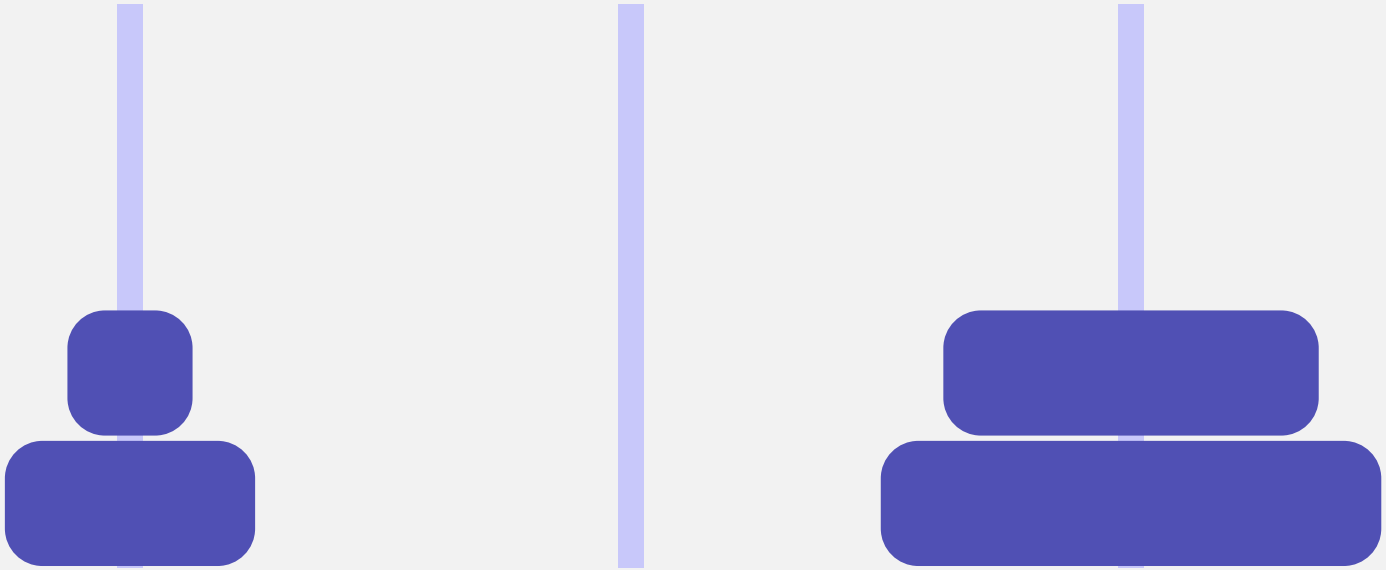
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!

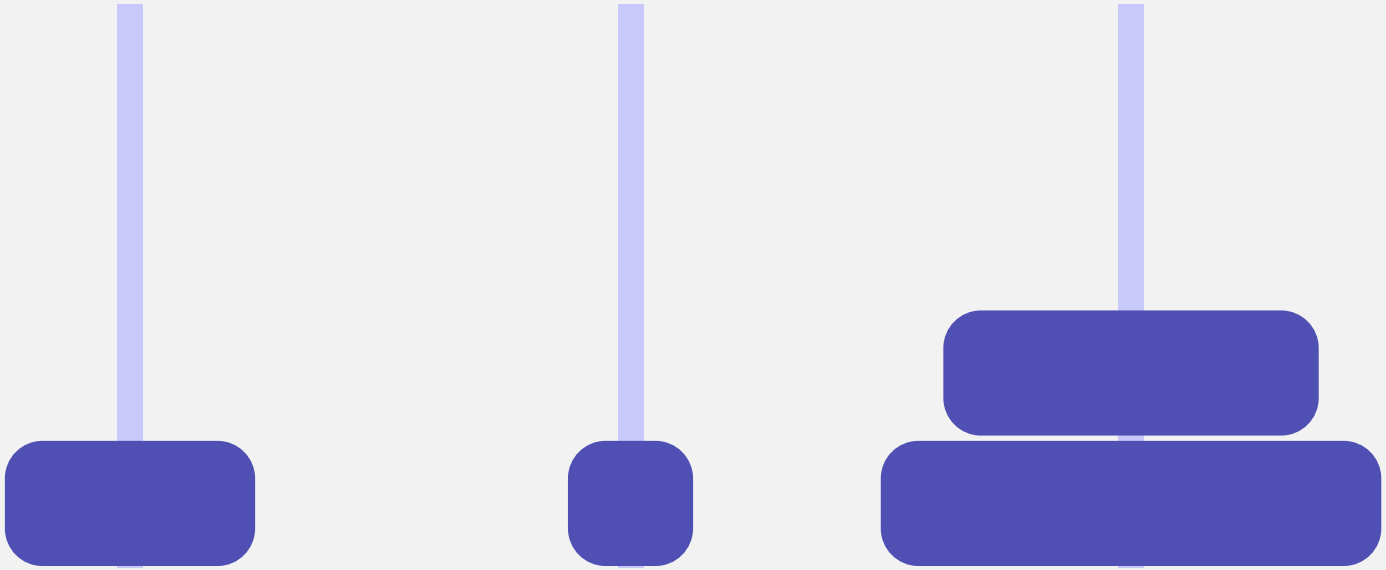


Die Türme von Hanoi - So gehts!

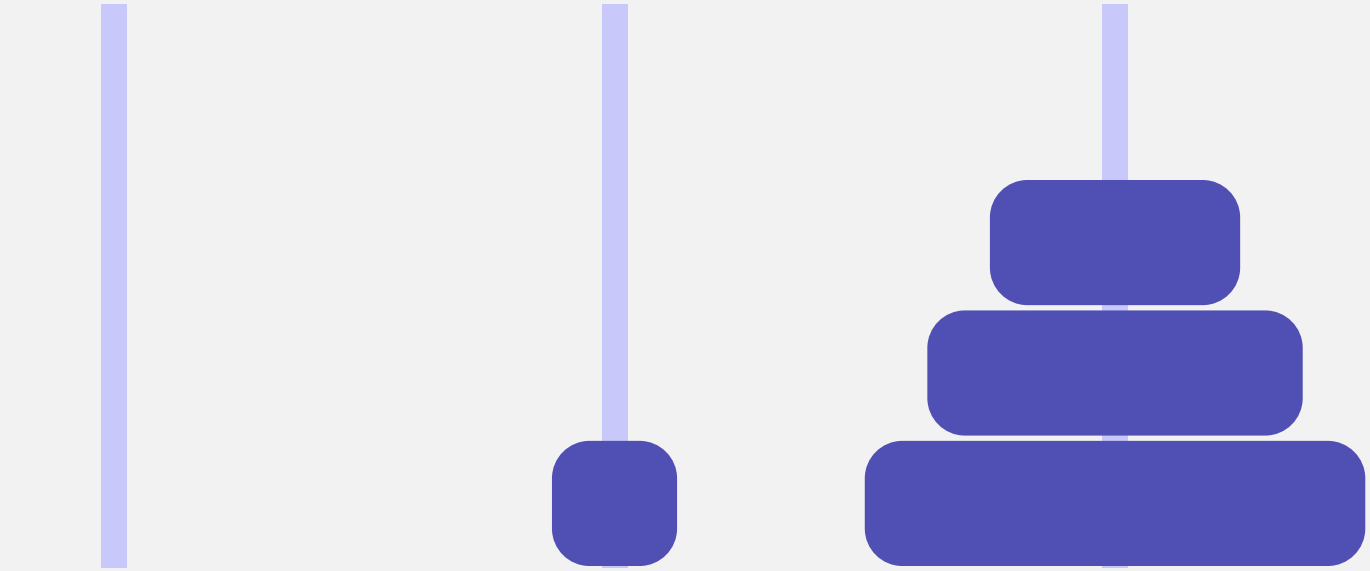


Links Mitte Rechts

Die Türme von Hanoi - So gehts!



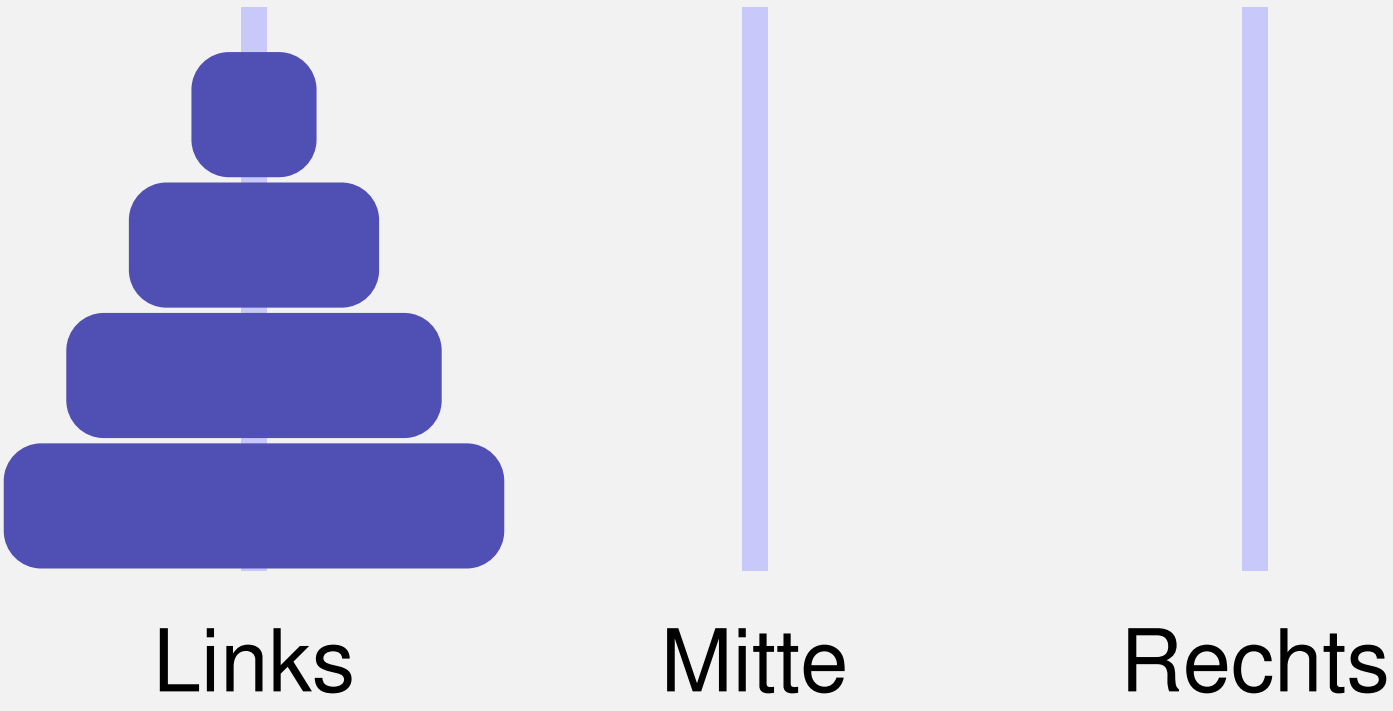
Die Türme von Hanoi - So gehts!



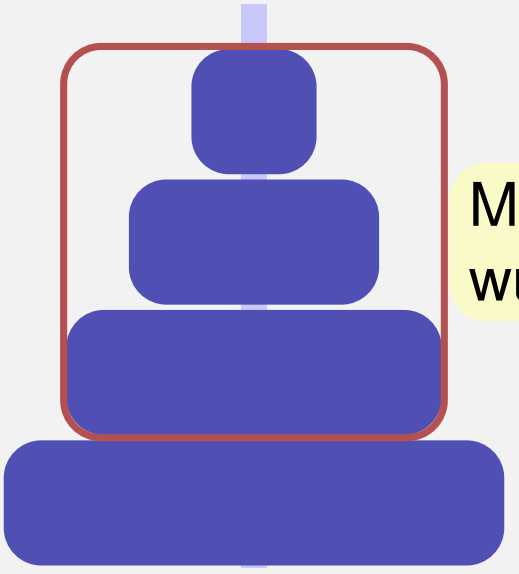
Die Türme von Hanoi - So gehts!

Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz

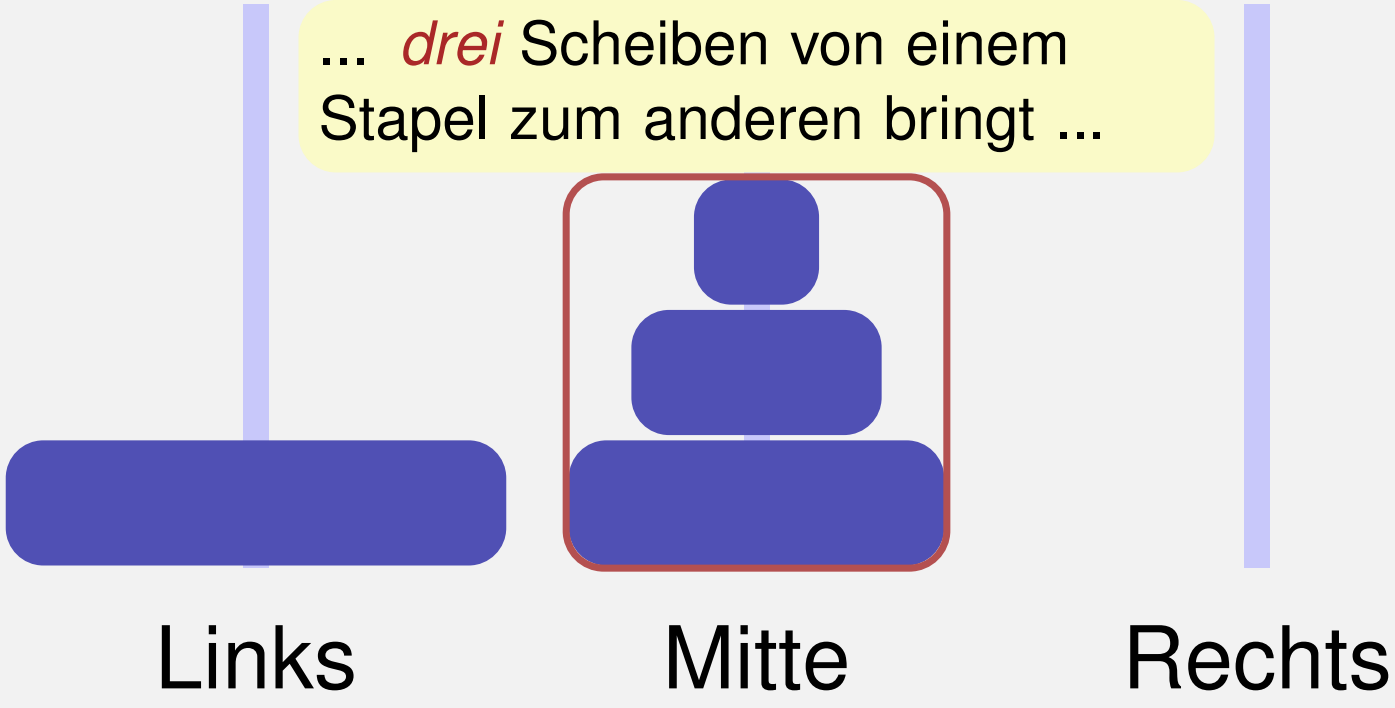


Mal *angenommen*, wir wüssten wie man ...

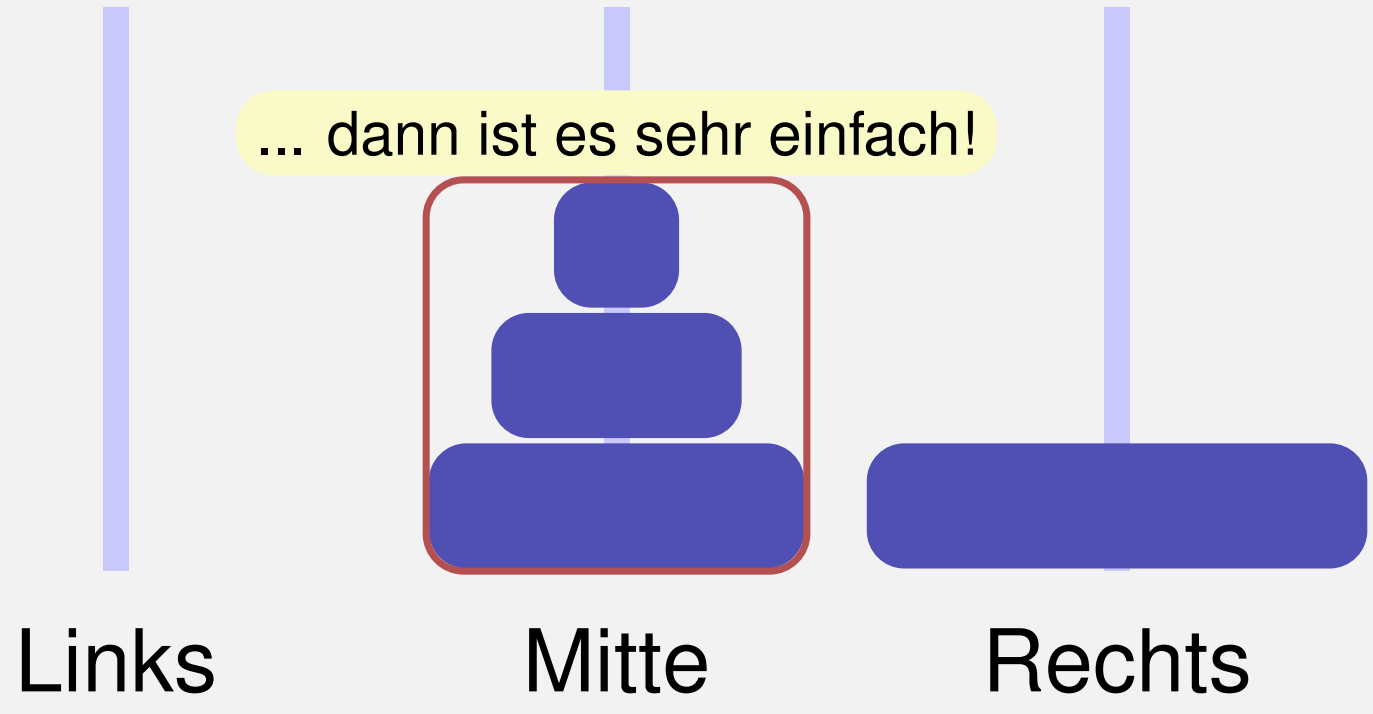
Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

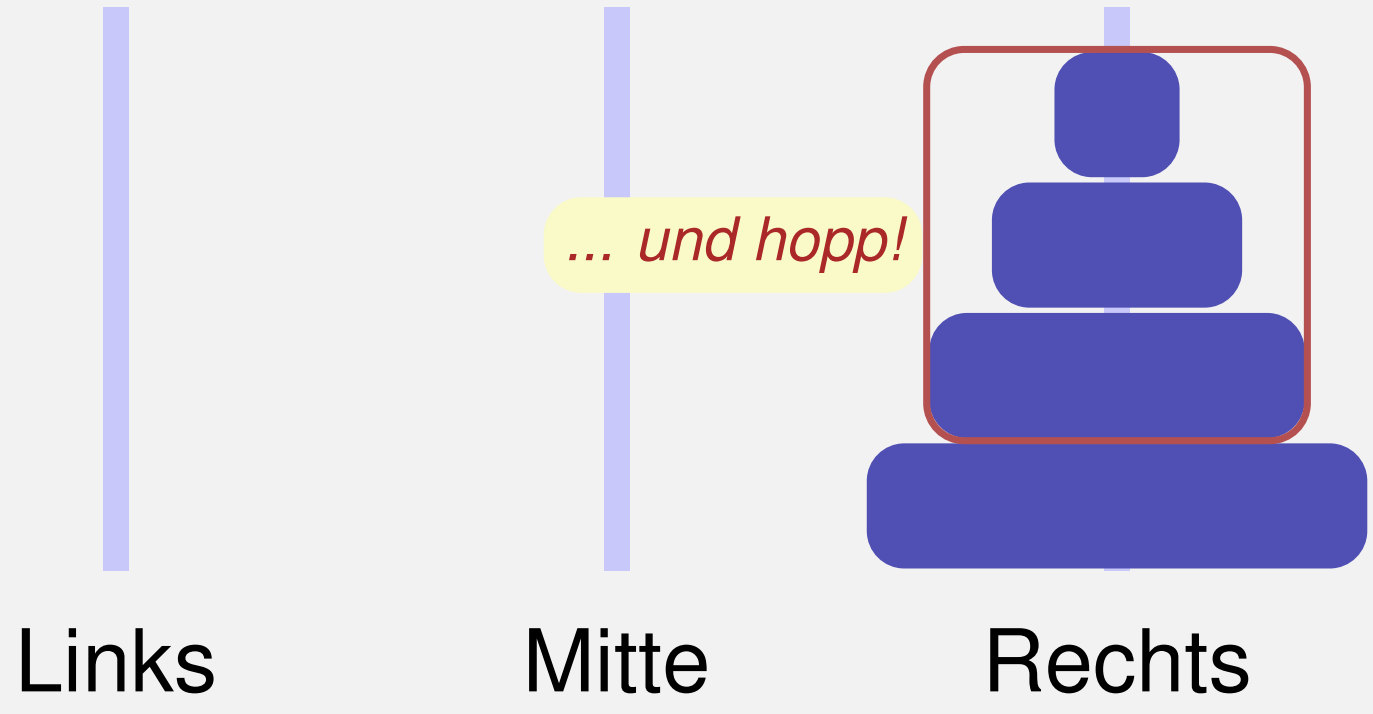
... *drei* Scheiben von einem Stapel zum anderen bringt ...



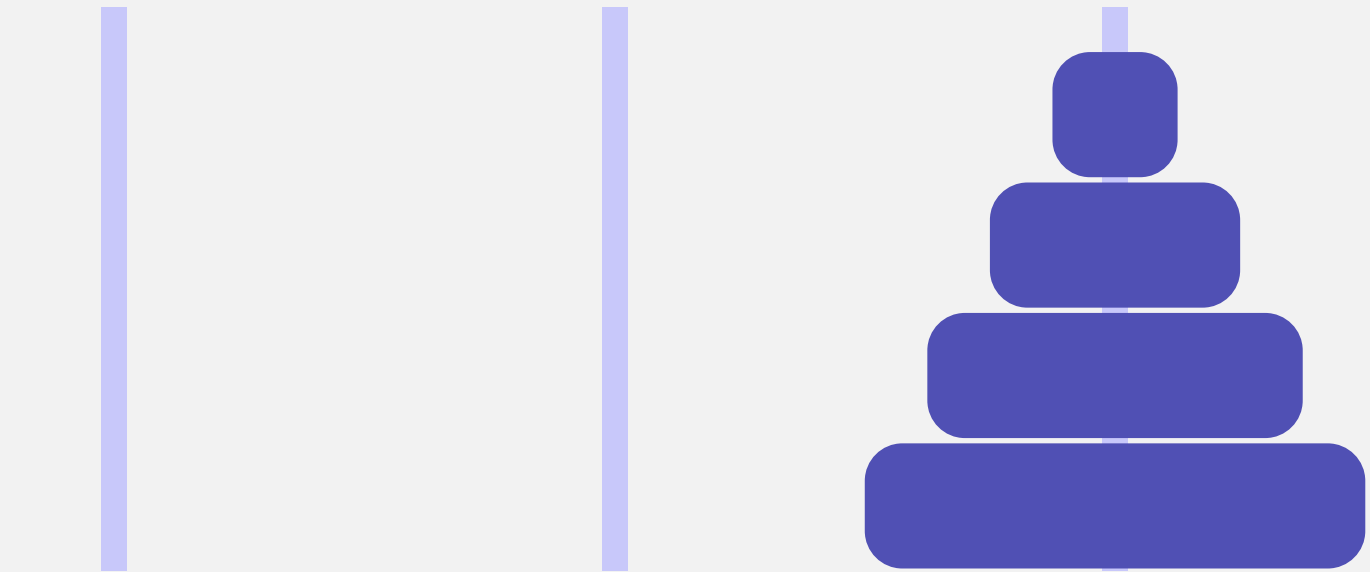
Die Türme von Hanoi - Rekursiver Lösungsansatz



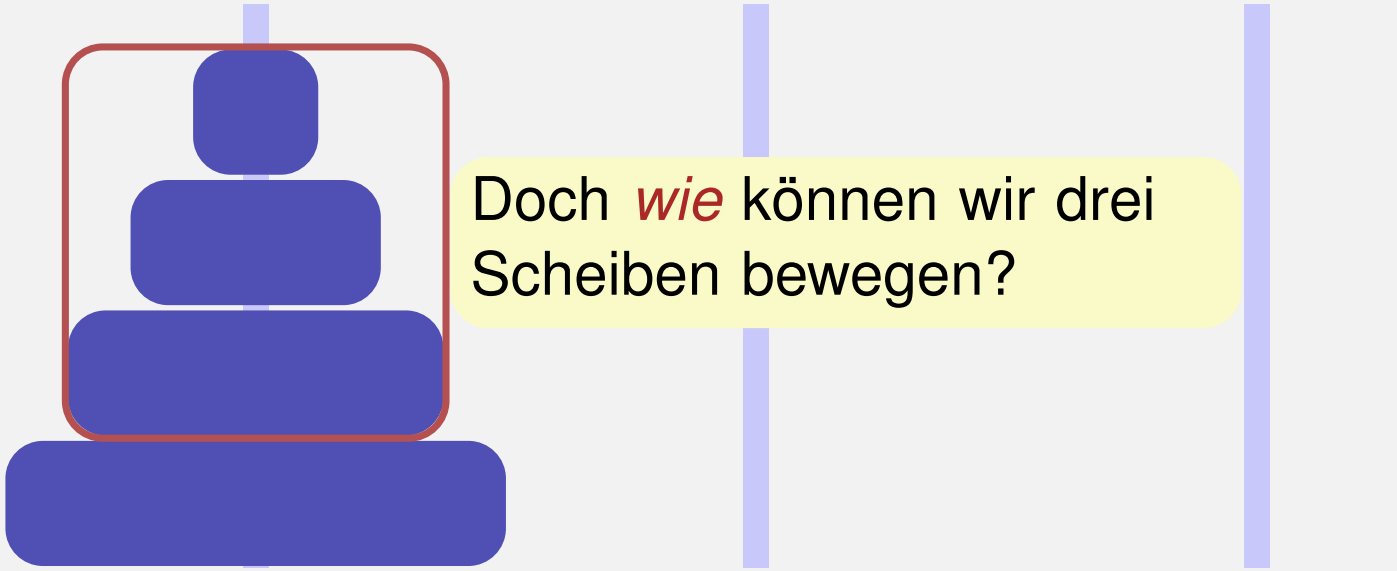
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



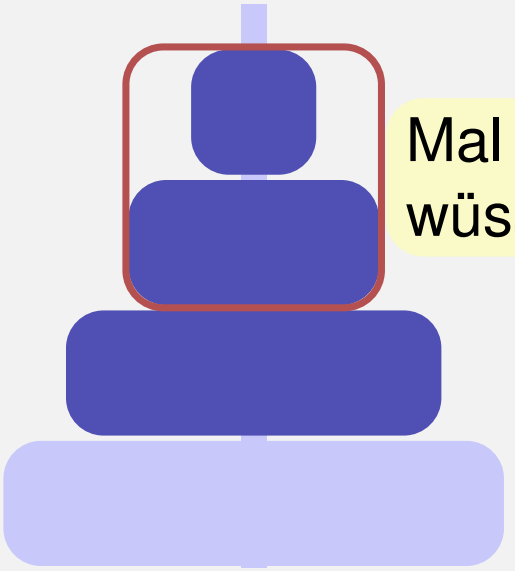
Die Türme von Hanoi - Rekursiver Lösungsansatz



Doch *wie* können wir drei Scheiben bewegen?

Links Mitte Rechts

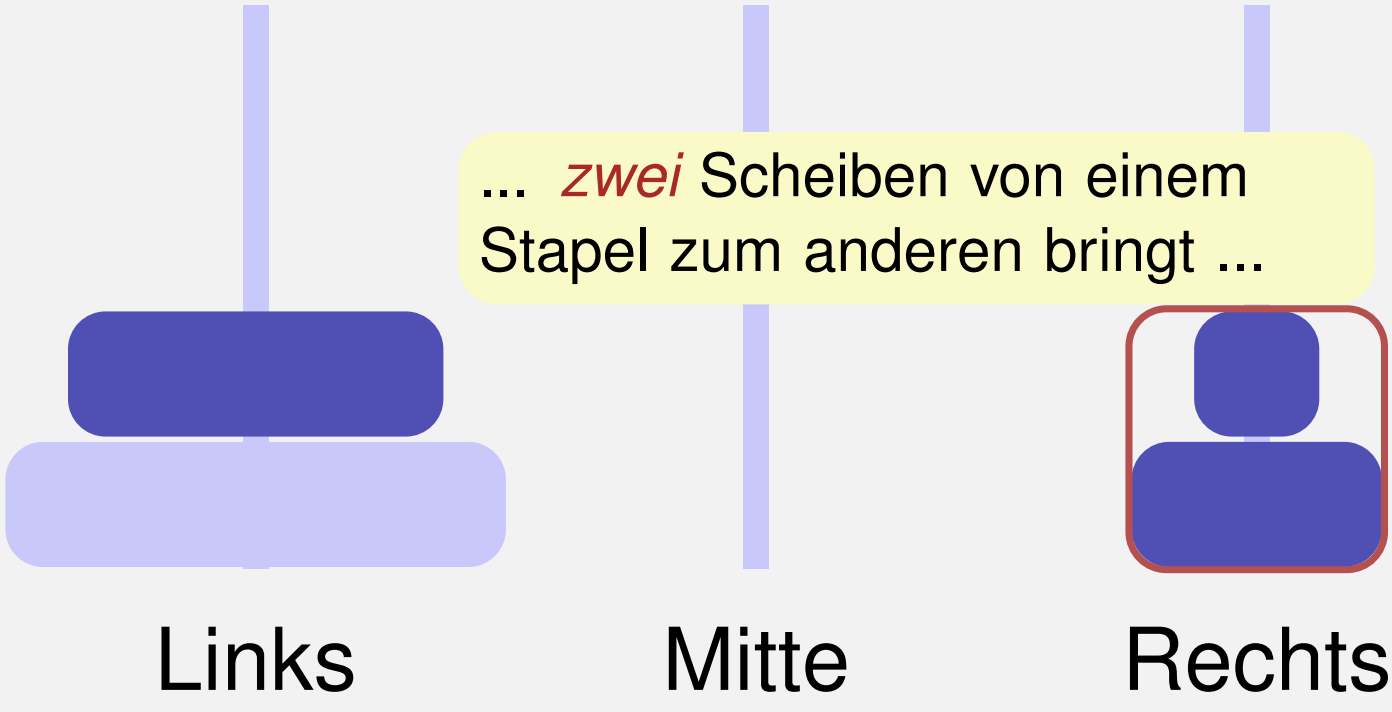
Die Türme von Hanoi - Rekursiver Lösungsansatz



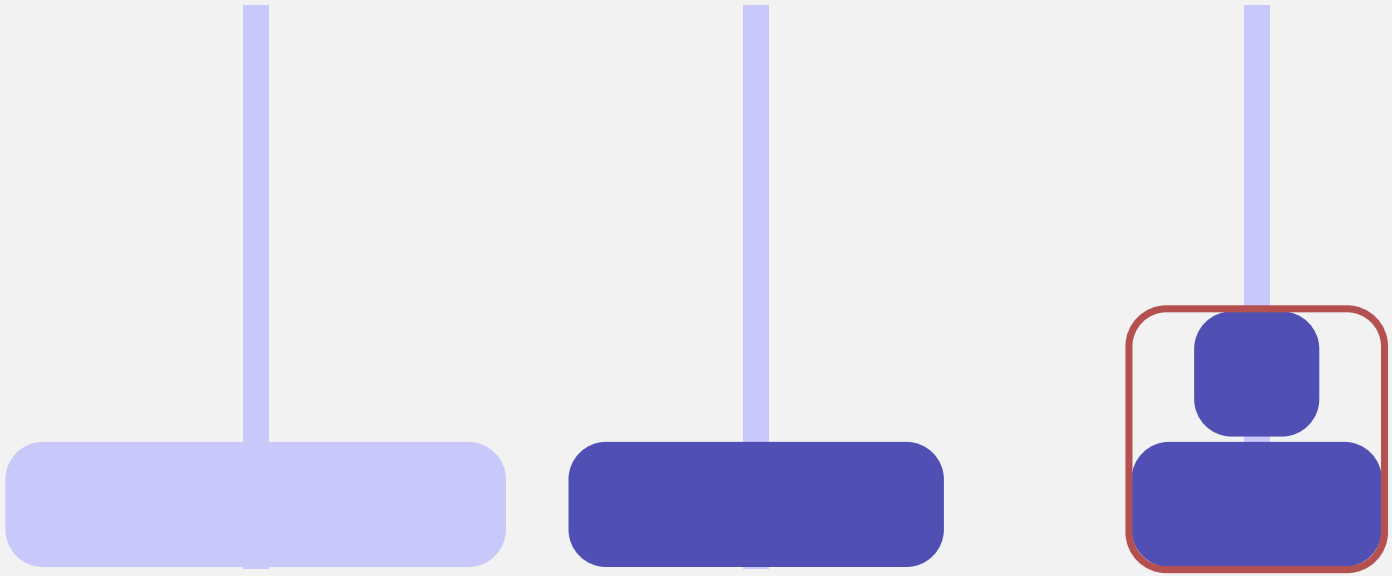
Mal *angenommen*, wir wüssten wie man ...

Links Mitte Rechts

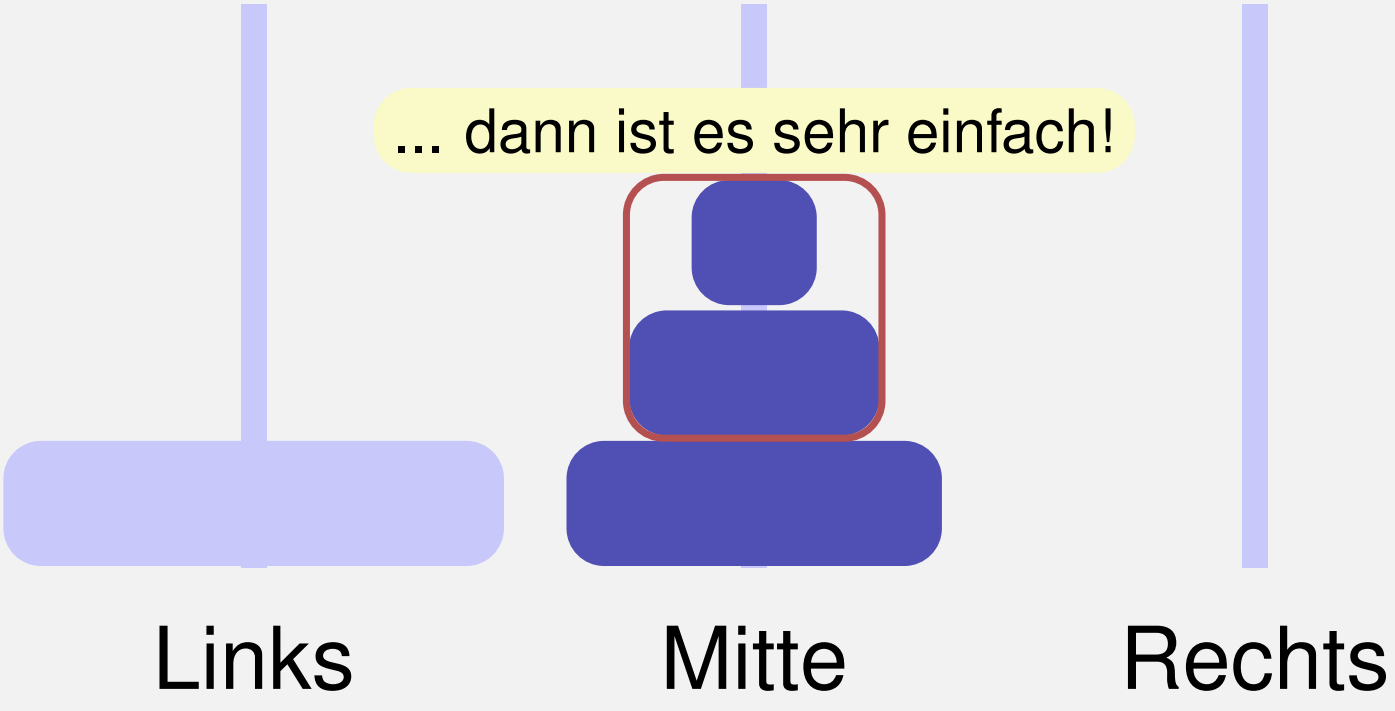
Die Türme von Hanoi - Rekursiver Lösungsansatz



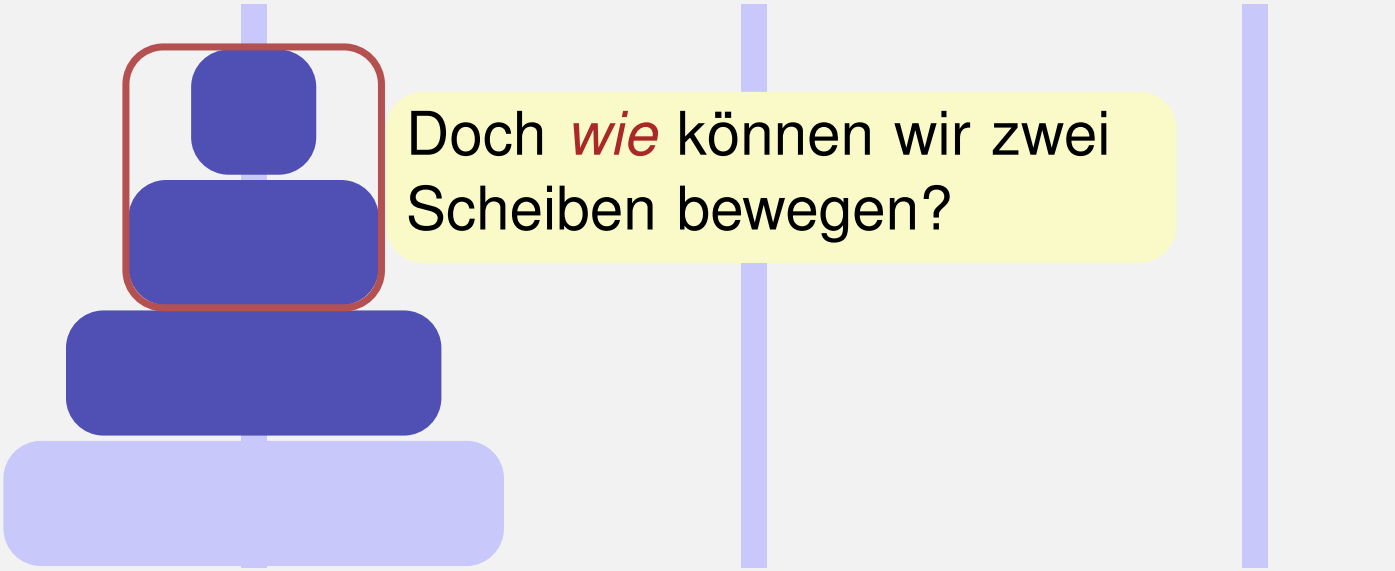
Die Türme von Hanoi - Rekursiver Lösungsansatz



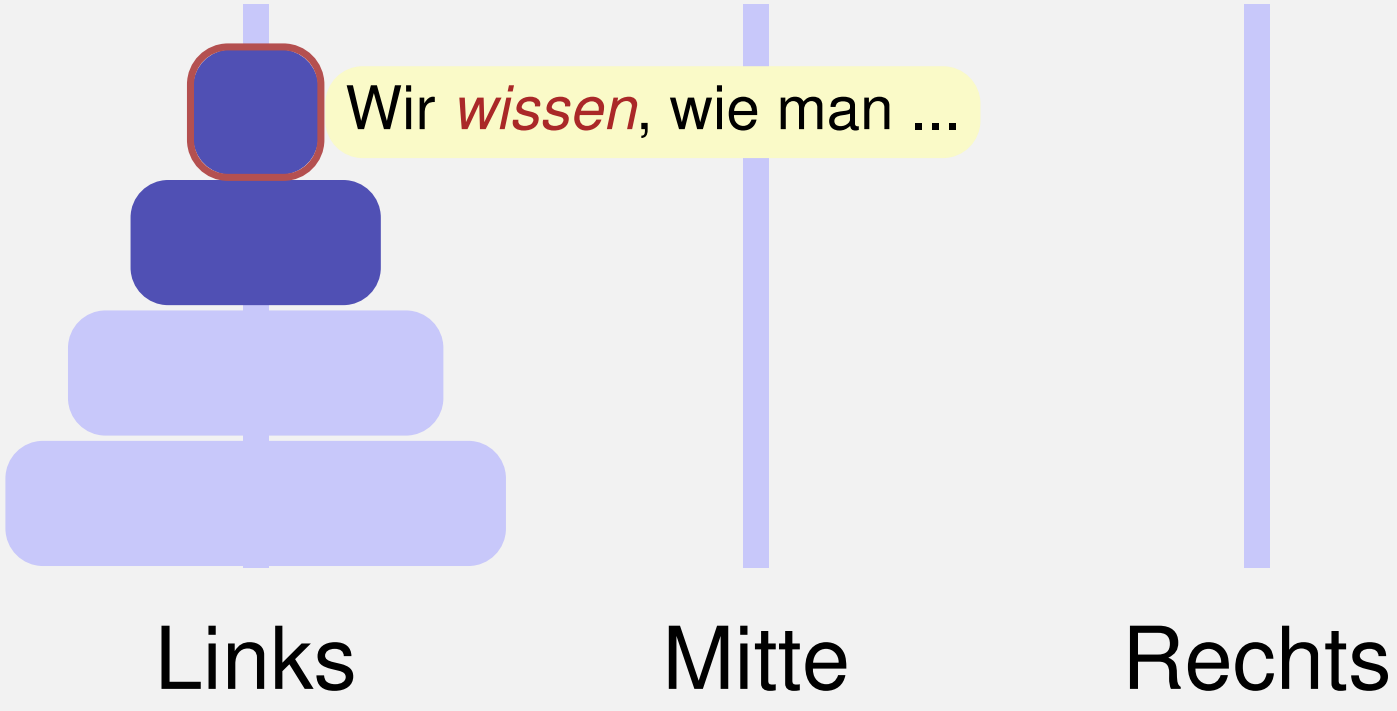
Die Türme von Hanoi - Rekursiver Lösungsansatz



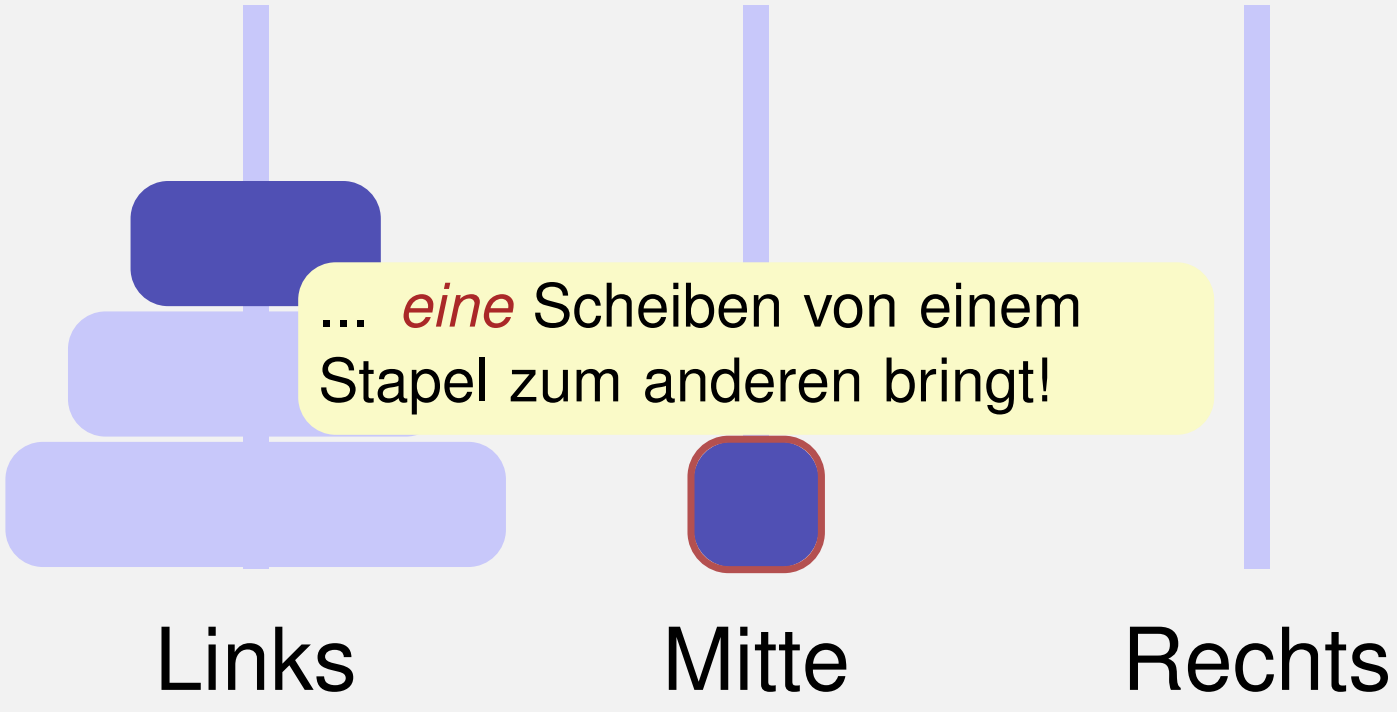
Die Türme von Hanoi - Rekursiver Lösungsansatz



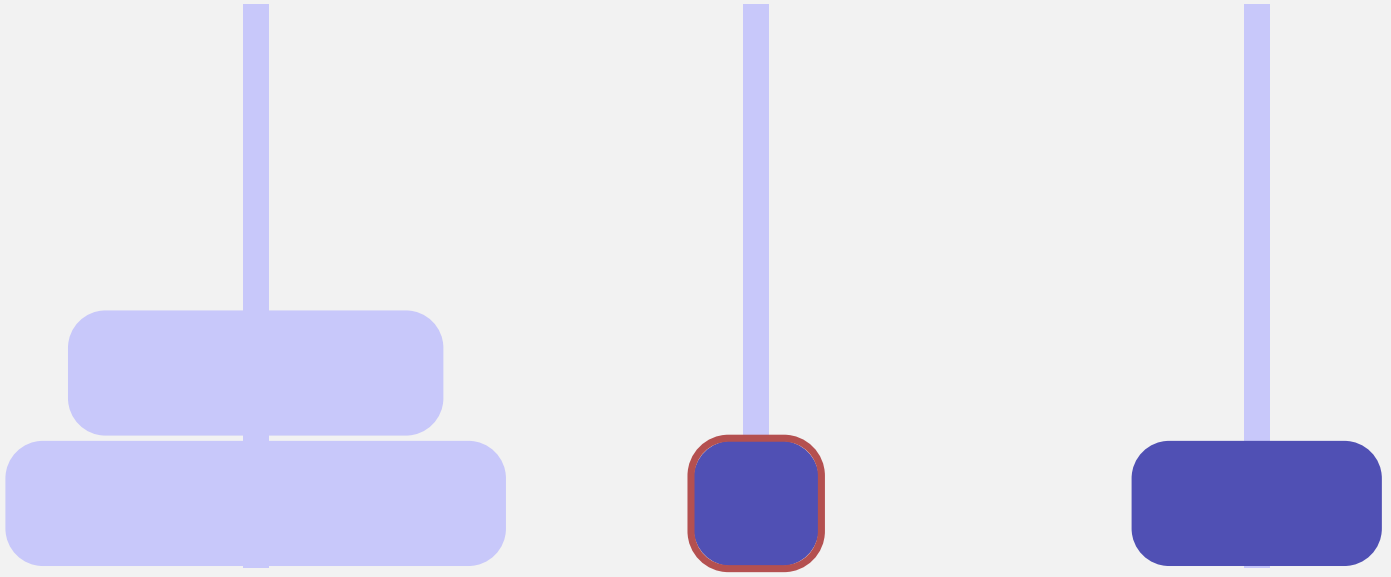
Die Türme von Hanoi - Rekursiver Lösungsansatz



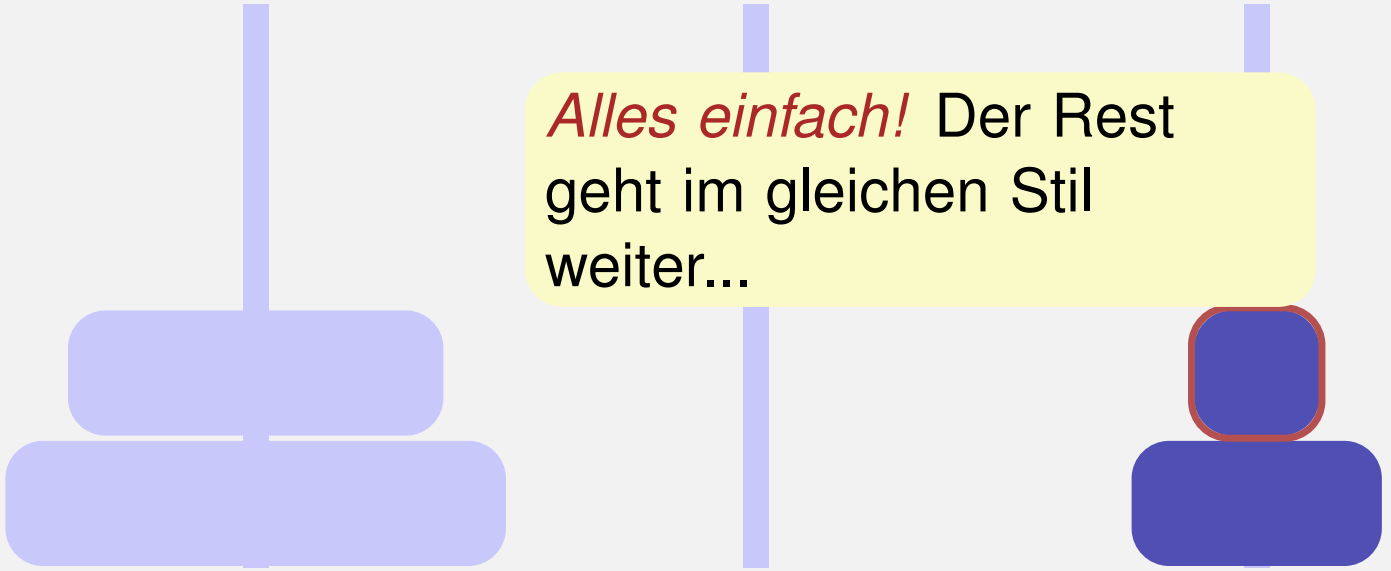
Die Türme von Hanoi - Rekursiver Lösungsansatz



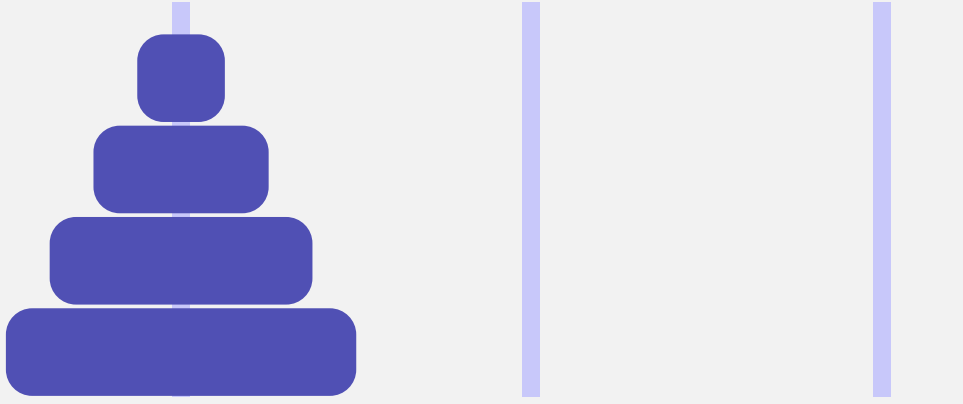
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Code



Bewege 4 Scheiben von `left` nach `right` mit Hilfsstapel `middle`:

```
move(4,"left","middle","right")
```

Die Türme von Hanoi - Code

`move(`n`,` src`,` aux`,` dst`)` $\Rightarrow$

1. Bewege die obersten $n-1$ Scheiben von src nach aux mit Hilfsstapel dst:
   `move(`$n-1$`,` src`,` dst`,` aux`);`
2. Bewege 1 Scheibe von src nach dst
   `move(1,` src`,` aux`,` dst`);`
3. Bewege die obersten $n-1$ Scheiben von aux nach dst mit Hilfsstapel src:
   `move(`$n-1$`,` aux`,` src`,` dst`);`

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case



    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);


    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);

    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case ('move' the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Die Türme von Hanoi - Code Alternative

```
void move(int n, const string &src, const string &aux, const string &dst){
    // base case
    if (n == 0) return;

    // recursive case
    move(n-1, src, dst, aux);
    std::cout << src << " --> " << dst << "\n";
    move(n-1, aux, src, dst);
}


int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Videoempfehlung

▸ Towers of Hanoi: A Complete Recursive Visualization

▸ 5 Simple Steps for Solving Any Recursive Problem

Versucht insbesondere das Konzept von *Recursive Leap of Faith* nachzuvollziehen. Das ist quasi die Induktionsannahme bei einem Induktionsbeweis aus der Mathematik

Übung ”Sequence Permutations”

- öffnet ”Sequence Permutations” auf [code]expert
- überlegt euch, wie ihr das Problem mit Stylus und Tablet angehen würdet
- programmiert eine (rekursive) Lösung (optional in Gruppen)
- Warnung: das wird *nicht einfach*

Lösung zu ”Sequence Permutations”

```
int main () {

  std::vector<int> sequence;

  // Read input.
  int num;
  std::cin >> num;
  while (num != -1) {
    sequence.push_back(num);
    std::cin >> num;
  }

  // A vector for tracking the current permutations.
  std::vector<int> permutation;
  permute(sequence, permutation);

  return 0;
}
```

Lösung zu ”Sequence Permutations”

```
// POST: Returns true iff element is in the permutation.
bool is_used(int element, std::vector<int>& permutation) {
  for (unsigned int i = 0; i < permutation.size(); ++i) {
    if (permutation.at(i) == element) {
      return true;
    }
  }
  return false;
}

// POST: Print the permutation to standard output.
void print_permutation(std::vector<int>& permutation) {
  for (unsigned int i = 0; i < permutation.size(); ++i) {
    std::cout << permutation.at(i) << " ";
  }
  std::cout << std::endl;
}
```

Lösung zu ”Sequence Permutations”

```
// POST: Prints out all possible permutations of the given sequence.
void permute(std::vector<int> sequence, std::vector<int> permutation) {
  if (sequence.size() == permutation.size()) {
    // We have a full permutation, just output it.
    print_permutation(permutation);
  } else {
    // Try all unused elements of the sequence.
    for (unsigned int i = 0; i < sequence.size(); ++i) {
      int element = sequence.at(i);
      if (!is_used(element, permutation)) {
        permutation.push_back(element);
        permute(sequence, permutation);
        permutation.pop_back();
      }
    }
  }
}
```

Fragen/Unklarheiten?

Tipps für [code]expert

E8:T1 Vector and matrix operations

- Achtet auf die Programmstruktur und befolgt die Task Description diesbezüglich genau
- Verwendet `using` um das Programm übersichtlicher zu machen
- Vergesst `//commenting`, `&` Referenzen und `const` nicht!

E8:T2 Decode binary NZZ front page

- *nicht einfach*
- ohne Summary kaum machbar
- fragt früh genug per Mail konkret nach
- Aufteilung in Subprobleme/Funktionen enorm wichtig um Überblick zu bewahren

E8:T3 Recursive function analysis

- Skizzen anfertigen hilft sehr

Allgemeine Fragen?

Bis zum nächsten Mal

- ☐ macht eure Hausaufgaben
- ☐ bleibt gesund