

Lesson 13

IFMP

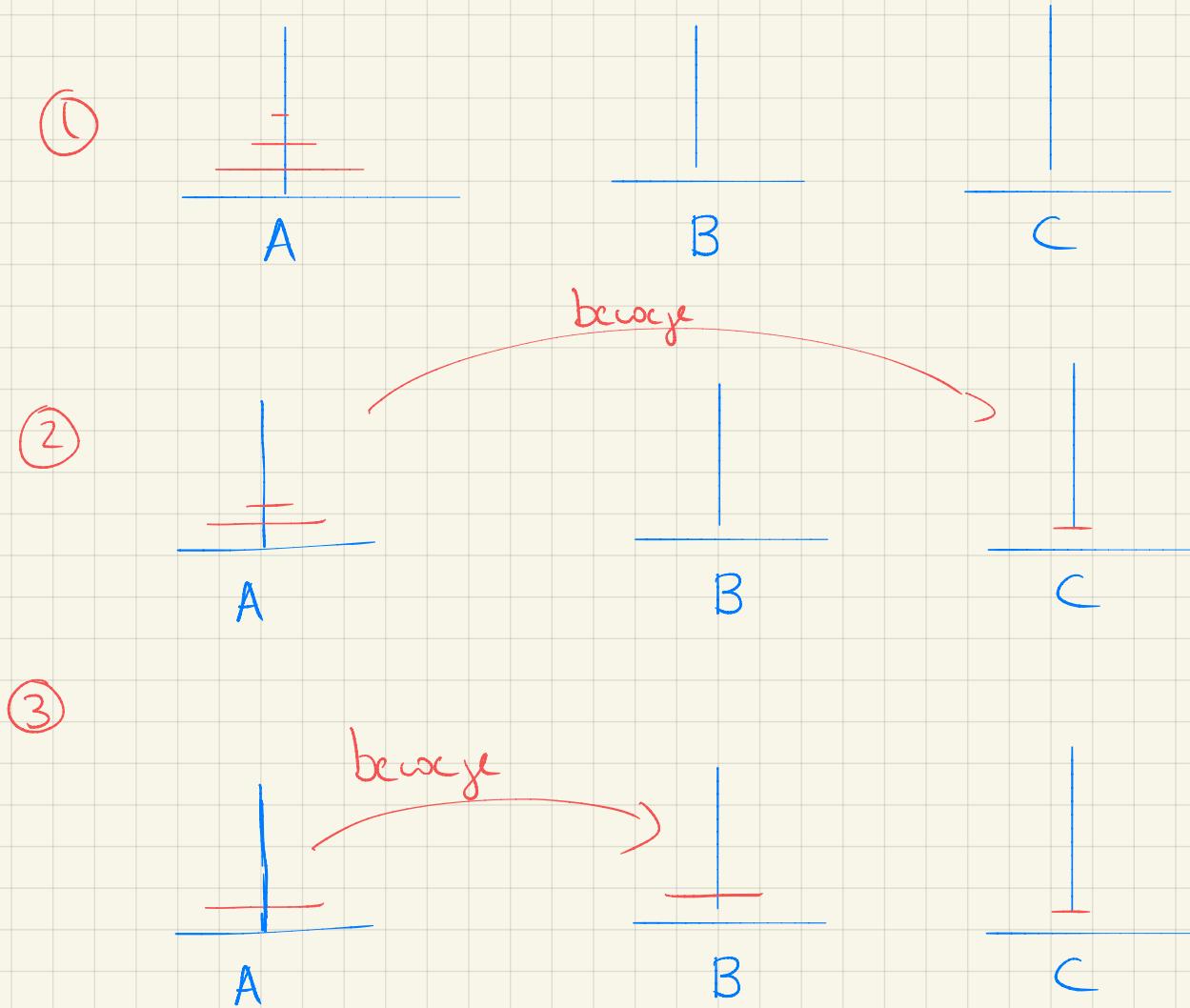
Main goals

- Explain Towers of Hanoi
- Do a better job
- Lesson Plan Week 13

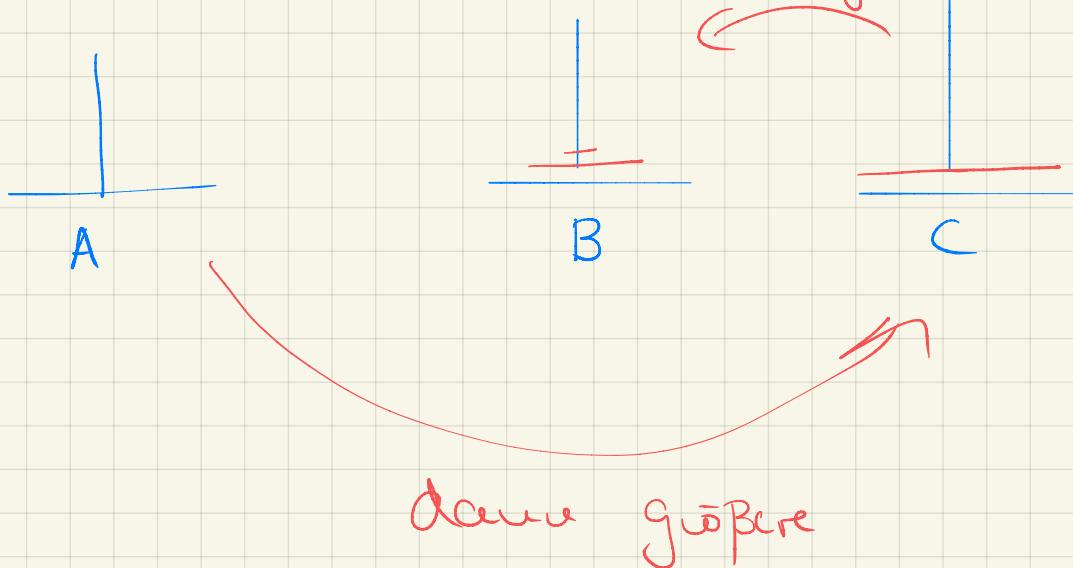
Towers of Hanoi

3 Stäbe A, B, C und u gelochte Scheiben. Das Ziel liegt darin von Anach C zu kommen ohne eine größere Scheibe über eine kleinere zu legen!

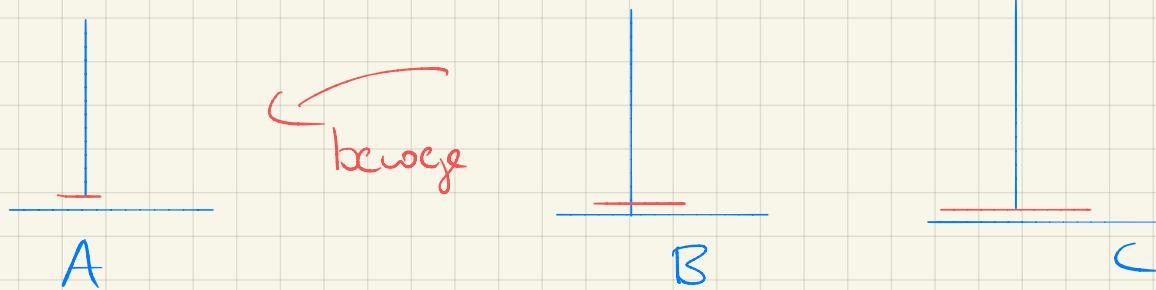
Kleines Beispiel $n=3$



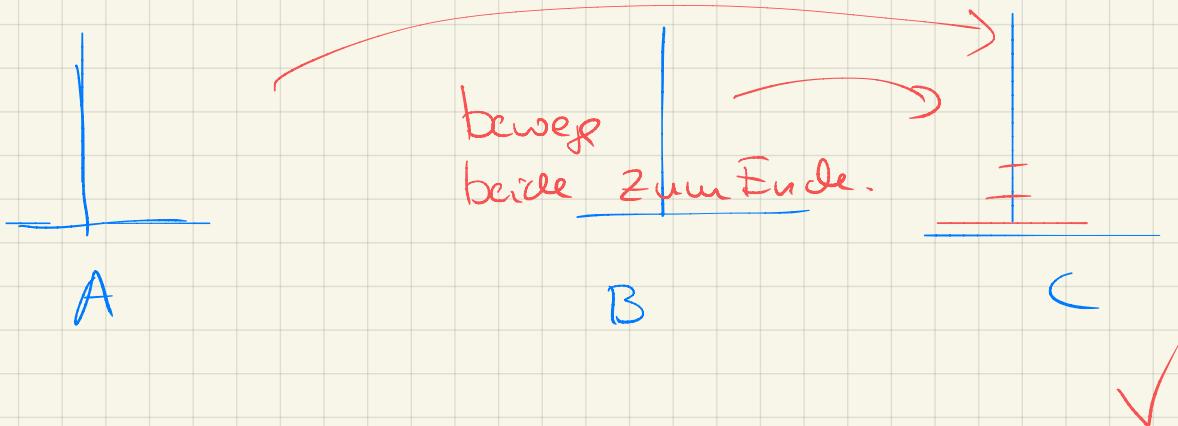
④



⑤



⑥



Man überträgt dieses Prinzip im allgemeinen - Induktiv stellt man fest: Die Schritte sind immer gleich!

- ① Bewege $n-1$ auf den Stapel B
- ② Bewege den größten Unterknopf auf den Stapel C
- ③ Bewege die $n-1$ von Stapel B auf C.

Vergleiche mit dem Code ⚡

```
void move(int n, const std::string &src, const std::string &aux, const std::string &dst) {
    if (n == 1) {
        // base case ('move' the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, aux, dst);   ← beweg n-1 auf aux
        move(1, src, aux, dst);    ← bewege unterster auf dst
        move(n-1, aux, src, dst);  ← bewege n-1 von aux auf dst.
    }
}
```

} wichtig zu tun!

Satz Die Anzahl der Züge bei einem Aufruf

bei move bei festem n ist $2^n - 1$.

Beweis

Challenge Task! Solution 26.1.2012 Prüfung

VMP Verzeichnis

18:08 Tue 10. Dec vmp.ethz.ch

3 of 4

Aufgabe 5.
Jeder Aufruf von `hanoi` für $n > 0$ Scheiben verursacht zwei Aufrufe von `hanoi` für $n - 1$ Scheiben und einen Zug. Also gilt

$$\begin{aligned} T(0) &= 0, \\ T(n) &= 2T(n-1) + 1, \quad n > 0. \end{aligned}$$

Behauptung: $T(n) = 2^n - 1$.

Beweis: Das gilt für $n = 0$ (Induktionsanfang). Sei nun $n > 0$, und nehmen wir an, die Behauptung gelte für $n-1$ (Induktionsannahme). Dann können wir (Induktionsschritt) wie folgt schliessen.

$$T(n) = 2T(n-1) + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1.$$

Punktvergabe. +10 Punkte für eine korrekte Rekursionsgleichung. Maximal +5 Punkte, falls es Fehler gibt (+1 vergessen etc.).

+5 Punkte für eine korrekt geratene Lösung (oder eine falsche Lösung, die dafür zur Rekursionsgleichung passt).

Delete operator

Wenn ihr new verwendet, dann gilt ihr fordert neuen Speicher auf dem Heap an! Konkret gilt dann auch:



Ihr fordert neuer Speicher irgendwo im Speicher.

Delete löscht diesen Speicher wieder!! Dann dies wird nicht manuell gemacht, sondern durch das delete!

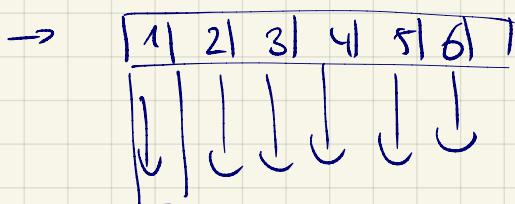
Destruktör

- Destruktör wird in einer Klasse durch `~Klassename()` beschrieben.
- Analog zum Konstruktör:
 - } Konstruktur wird gerufen, wenn Objekt erstellt wird (new)
 - } Destruktör wird gerufen, wenn Objekt gelöscht wird (delete)
- call lebt ~ im Destruktör call.

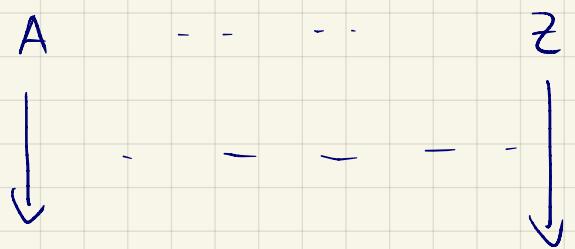
Datstrukturen (u. Wahrheit nach einer)

1. Hash Table

→ Ähnlich wie Linked List. ⇒ statt nur list "array" von Linked-Lists (so implementiert.)



Bsp. Telefonbuch!



"Befreig erwartet bei"

2. Exam question "HasleTable"

Box Destructor

Implementieren verschieden Constructors. Es gibt den copy-constructor, assignment constructor, den Destruktor und den Konstruktor.

1. Destruktor: Spezifiz. was pass. wenn Objekt zwst. wird

2. Konstruktor Siche früher

3. Assignment: Spezifiz. was passiert bei Zuweisung passiert?

4. Copy: Beim Kopieren des Objekts!

Wie geben Box(const Box& other) → copy constructor: const ref wichtig!
übergeben const ref an Box

Assignment constructor operator overloading

Box & operator =
↑
R wichtig.

Warum solche Konstrukte

→ Effizienter Speicher

→ mehr Flexibilität

Box class

Prop: int * ptr;

Methods: Box (const Box& other)

copy const

Box & oper. = ...

assignment

~Box();

destructor

Copy constructor

- Initialisiere pointer auf int mit selben Wert wie other
- equals

Assignment

- (idea: $\ast\text{ptr} = \ast\text{other}.\text{ptr}$) \leftarrow weise das zu \rightarrow
return $\ast\text{this};$

Destruktör

[delete ptr;]

\rightarrow gute Aufgabe! Ausgabe testen und Ausgeben.

Mach für Destruktör vor.

Klausur Hash Table

```
struct contact {  
    std::string name; // Contact's name  
    unsigned int number; // Contact's phone number  
    contact* next; // Another contact (or nullptr)  
};
```

Werde vectors of contacts, implementiere insert und delete funktionen.

Insert

- \rightarrow Definiere Zahl korrekt unsigned int
- \rightarrow füghinzu

Delete

- \rightarrow find target list
- \rightarrow check all cases like linked list!

9 Programme: Hash-Tabelle (16 Punkte)

Eve entwickelt ihre eigene Kontaktlisten-App um die Telefonnummer ihrer Freunde zu verwalten. In ihrer App werden Kontakte als struct contact repräsentiert.

```
struct contact {
    std::string name; // Contact's name
    unsigned int number; // Contact's phone number
    contact* next; // Another contact (or nullptr)
};
```

Eve möchte Kontakte so abspeichern, dass der zu einer Telefonnummer gehörende Kontakt effizient gefunden werden kann (es darf angenommen werden, dass keine zwei Kontakte die selbe Telefonnummer haben). Dafür nutzt Eve eine Hash-Tabelle als Datenstruktur, die die Kontakte intern in einem Vektor der Grösse n ablegt: Jeder Kontakt wird auf eine natürliche Zahl i abgebildet, berechnet als „Telefonnummer modulo n “, anschliessend wird der Kontakt im Vektor an Index i abgelegt.

Zum Beispiel: für $n = 7$ erhält die Telefonnummer 70 00 03 den Index 3, denn $700003 \bmod 7 = 3$.

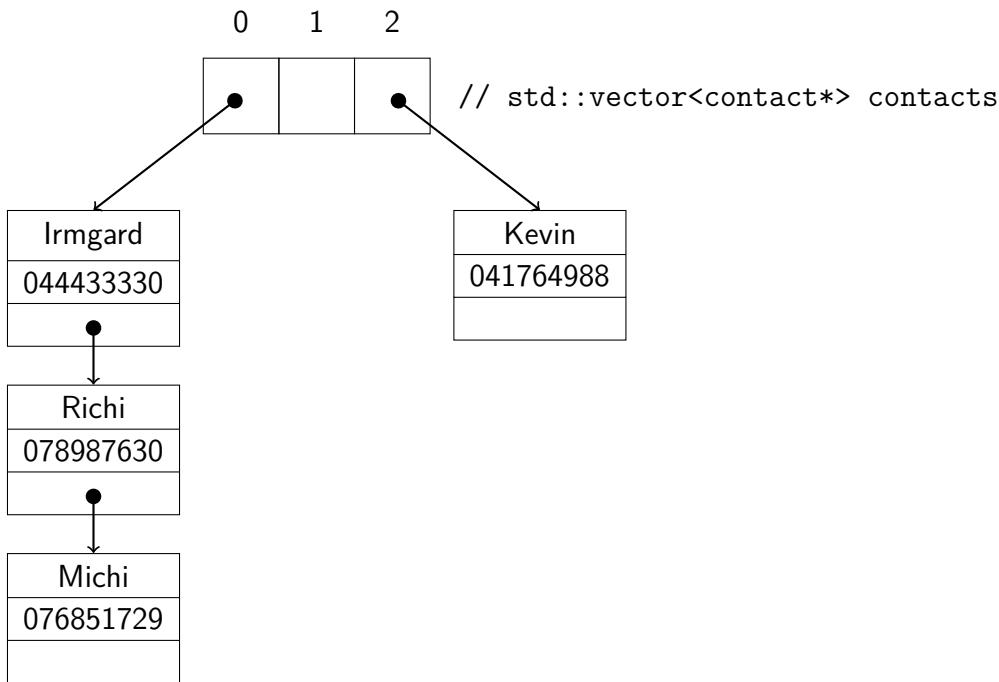
Da diese Berechnung verschiedene Telefonnummern auf den selben Index abbilden kann, hat Eve sich dazu entschieden, diese als verkettete Liste zu verknüpfen, wie in folgendem Diagramm illustriert.

Eve decided to develop her own contact list app for managing her friends' phone numbers. In the app, contacts are represented by a struct contact.

Eve wants to store the contacts such that, given a telephone number, she can efficiently find the corresponding contact (you can assume that no two contacts have the same phone number). For that, she uses a hash table data structure that internally stores the contacts in a vector of size n : each contact is mapped to a natural number i by computing “phone number modulo n ”, and then the contact is stored at index i of the vector.

For example: given $n = 7$, the index for phone number 70 00 03 is 3 because $700003 \bmod 7 = 3$.

Since this computation can map different phone numbers to the same index, Eve decided to connect them as a linked list, as illustrated in the diagram below.



Vervollständigen Sie Eves unten stehende Implementierung.

Fill the code blanks below to finish Eve's implementation.

```
#include <vector>
#include "contact.h" // Struct contact, as shown above

class contact_list {
    std::vector<contact*> contacts;

public:
    // POST: Initialises vector 'contacts' with 'size' null-pointers
    contact_list(unsigned int size):
        contacts(std::vector<contact*>(size, nullptr)) {}

    // POST: The new contact was prepended to the contact list
    void insert(std::string& name, unsigned int number) {
        unsigned int idx = number % contacts.size();
        contacts[idx] = new contact{name, number, contacts[idx]};
    }
}
```

```
// POST: Returns true if a contact with the phone number was
//       found and removed, and false otherwise
bool remove(unsigned int number) {
    unsigned int idx = number % contacts.size();

    contact* prev = nullptr;
    contact* curr = contacts[idx];

    while (curr != nullptr && curr->number != number) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == nullptr) return false;

    if (curr == contacts[idx]) {
        contacts[idx] = curr->next;
    } else {
        prev->next = curr->next;
    }
    delete curr;
    return true;
}
};
```