ZALG 5. cvičení

Metoda rozděl a panuj

- Jedna ze základních metod tvorby algoritmů
- Základní myšlenka:
  - Potřebujeme zpracovat množinu V
  1. **ROZDĚL:** Rozdělíme tuto množinu na k disjunktních podmnožin
  2. **VYŘEŠ:** Každou množinu zpracujeme zvlášť
  3. **SPOJ:** Dílčí výsledky spojíme a dostaneme řešení pro celou množinu V

Rekurze – metoda vyřeš

- Problém zpracování dílčích podmnožin je stejného typu jako zpracování množiny V jen s méně údaji → REKURZE

- Metoda vede na rekurzivní metodu se složitostí ve tvaru:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \text{ kde } T(1) = f(1)$$

  - a - je počet podúloh, na které je úloha rozdělena
  - $\frac{n}{b}$ - je velikost jedné podúlohy
  - $f(n) = O(n^c)$ – je časová složitost fází ROZDĚL a SPOJ

Problémy

1. Binární vyhledávání
2. Postavení binárního vyhledávacího stromu z uspořádaného pole
3. Hanojské věže

Binární vyhledávání (vyhledávání půlením intervalu)

- Máme **uspořádané** pole délky n
- Chceme v poli rychle vyhledat klíč (na jakém indexu se nachází)

- Řešení:
  - Počáteční interval $[l, r] = [0, n-1]$, najdeme prostřední prvek $s = \frac{l+r}{2}$
    - Nachází se na indexu *s* hledaný klíč? → Konec
    - Je prvek na indexu větší než hledaný klíč? → přejdeme na interval $[l, s-1]$
    - Je prvek na indexu menší než hledaný klíč? → přejdeme na interval $[s+1, r]$

→Algoritmus imituje vyhledávání v perfektním binárním vyhledávacím stromě uloženém v poli

Binární vyhledávání - implementace

- Možné implementace:
  1. Pomocí rekurze
  2. Iterativně (bez rekurze)

- Jaká je časová složitost?

Binární vyhledávání - složitost

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad \text{kde} \quad T(1) = f(1)$$

- a - je počet podúloh, na které je úloha rozdělena
- $\frac{n}{b}$ - je velikost jedné podúlohy
- $f(n) = O(n^c)$ – je časová složitost fází ROZDĚL a SPOJ

- a = 1
- b = 2
- $f(n) = O(1) = O(n^1) \rightarrow$ c = 0

Binární vyhledávání – Master Theorem

- a = 1
- b = 2
- c = 0

- $\rightarrow a = b^c$
- $\rightarrow$ typ B
- $O(n^0 \log_2 n) = O(\log_2 n)$

Nechť $a, b, c \in N$ a $f : N \to N$ je funkce, pro kterou platí $f(n) = O(n^c)$. $T(n)$ je neklesající posloupnost taková, že $\forall n : n = b^k, k \in N$ platí:

$$T(n) \leq aT(\frac{n}{b}) + f(n)$$

$$T(1) = f(1)$$

Potom

- je-li $a < b^c$, $\to T(n) = O(n^c)$ ... **typ A**
- je-li $a = b^c$, $\to T(n) = O(n^c log_b n)$ ... **typ B**
- je-li $a > b^c$, $\to T(n) = O(n^{log_b a})$ ... **typ C**

Postavení binárního vyhledávacího stromu z uspořádaného pole

- Máme uspořádané pole délky n
- Chceme postavit binární vyhledávací strom o optimální hloubce

- Řešení:
  - Chceme postavit strom z hodnot na indexech $\{\mathrm{l}, \dots, \mathrm{r}\} = \{0, \dots, n-1\}$
    - Do kořene stromu dáme prostřední prvek pole, $s = \frac{l+r}{2}$
    - Jeho levý podstrom postavíme z intervalu $\{\mathrm{l}, \dots, \mathrm{r}\} = \{l, \dots, s-1\}$
    - Jeho pravý podstrom postavíme z intervalu $\{\mathrm{l}, \dots, \mathrm{r}\} = \{s+1, \dots, r\}$

Postavení binárního vyhledávacího stromu

- $T(n) = 2T\left(\frac{n}{2}\right) + O(1)$
- a = 2 (počet podúloh)
- b = 2 (velikost jedné podúlohy)
- c = 0

- $\rightarrow a > b^c \rightarrow$ Typ C
- $O\left(n^{\log_b a}\right) = O\left(n^{\log_2 2}\right) = O(n)$

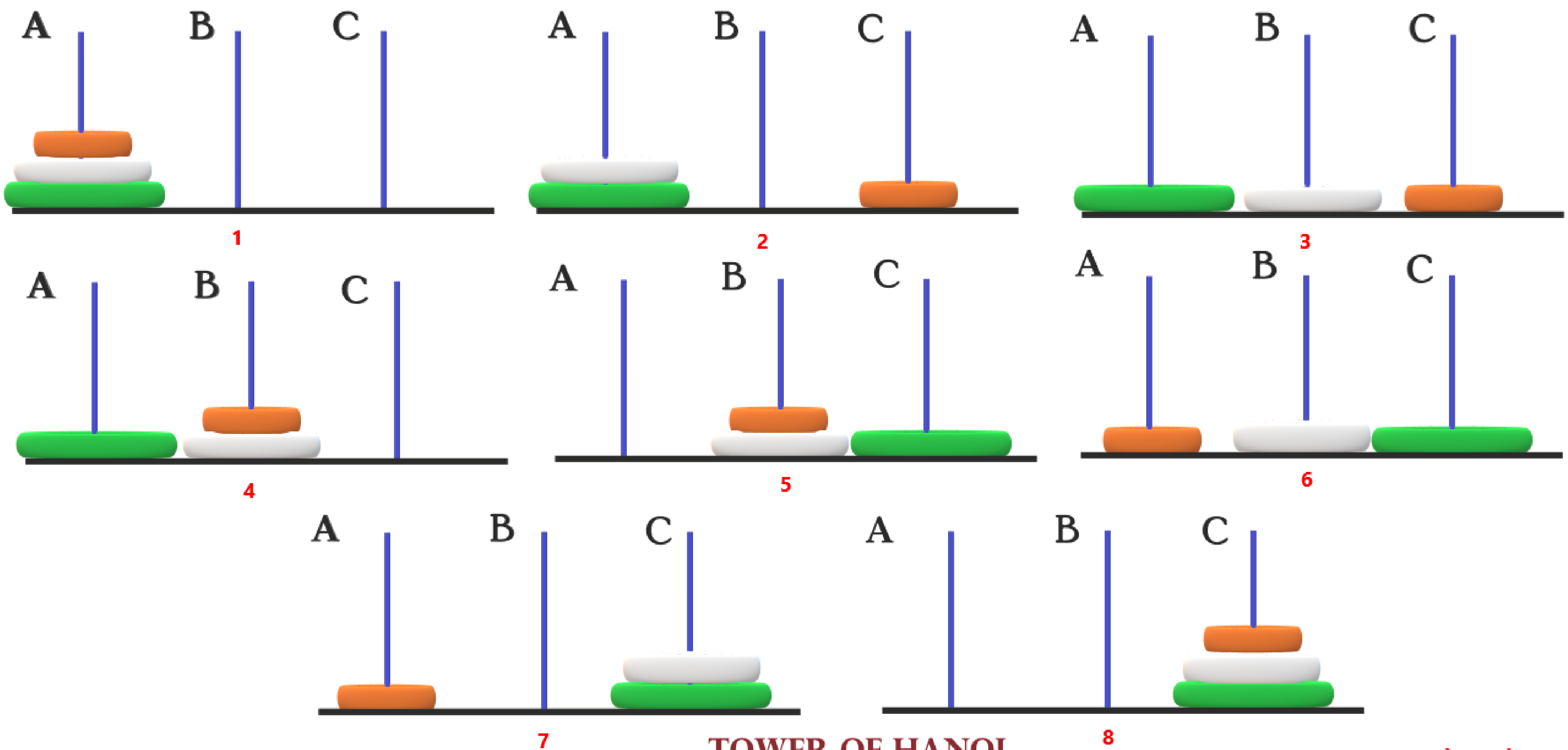
Implementace

- Strom zdrojový soubor: https://github.com/martinnovaak/ZALGcv
- Řešení:
  - 1) Rekurzivní
  - 2) Přes zásobník

Hanojské věže

- Máme 3 kolíky a n kotoučů o různých poloměrech

- **Začátek:** všechny kotouče jsou umístěné na 1. kolíku v pořadí od největšího po nejmenší (vždy menší na větším)
- **Cíl:** všechny kotouče jsou umístěné na 3. kolíku v pořadí od největšího po nejměnší



TOWER OF HANOI

Hanojské věže

- Pravidla přesouvání kotoučů:
  1. Vždy přesouváme jen jeden kotouč
  2. Není povoleno odložit kotouč mimo kolíky
  3. Není povoleno položit větší kotouč na menší

- Kolík tedy pracuje na principu LIFO (Last In, First Out). Jedná se tedy o zásobník. (Kvůli výpisu budeme ovšem kolíky implementovat přes std::vector, ovšem budem s nimi pracovat jako se zásobníky)

Hanojské věže – řešení metodou rozděl a panuj

- Dá se odpozorovat následující rekurentní řešení:
  1. Přesuň horních $n - 1$ disků na pomocný kolík
  2. Přesuň poslední n-tý disk na cílový kolík
  3. Přesuň $n - 1$ disků z pomocného kolíku na cílový

- POZN: Krok 1 a 3 vedou ne rekurzivní řešení. V 1. kroku se z kolíku číslo 3 stává pomocný kolík a v 3. kroku se z kolíku číslo 1 stává pomocný kolík.

Hanojské věže – rekurzivní řešení

```
void move_discs(int n, int source, int auxiliary, int destination) {
    if(n == 1)
        move_disc(source, destination);
    else
    {
        move_discs( n: n-1, source,  auxiliary: destination,  destination: auxiliary);
        move_disc(source, destination);
        move_discs( n: n-1,  source: auxiliary,  auxiliary: source, destination);
    }
}
```

Asymptotická složitost algoritmu

- Máme rekurentní vztah:

$$T(n) = 2 \cdot T(n-1) + 1$$
$$T(1) = 1$$

→ z DIM2 víme, že řešení je: $T(n) = O(2^n - 1) = O(2^n)$

Kdyby nám přesunutí jednoho kotouče trvalo 1 sekundu. Tak nám vyřešení hanojských věží s 64 kotoučema zabere celkem: 600 000 000 000 let

```
void solve_iteratively() {
    initialize();
    int source = 0, auxiliary = 1, destination = 2, total_moves = 1 << (number_of_discs - 1);
    if(number_of_discs & 1) {
        auxiliary = 2; destination = 1;
    }

    for(int i = 0; i < total_moves; i++) {
        switch (i % 3) {
            case 1:
                move_disc_legally(source, destination);
                break;
            case 2:
                move_disc_legally(source, destination: auxiliary);
                break;
            case 0:
                move_disc_legally( source: auxiliary, destination);
                break;
        }
    }
}
```