Rekurze

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 6

A0B36PR1 – Programování 1

Část 1 – Rekurze

Faktoriál

Obrácený výpis

Karel

Hanojské věže

Rekurze

Fibonacciho posloupnost

Část 2 – Příklady

Eratosthenovo síto

Rozklad na prvočinitele

Řazení

Část I

Rekurze

Obsah

Faktoriál

Obrácený výpis

Karel

Hanojské věže

Rekurze

Fibonacciho posloupnost

Výpočet faktoriálu

- Iterace

$$n! = n(n-1)(n-2)\ldots 2\cdot 1$$

- Rekurze

$$n! = 1 \text{ pro } n \le 1$$
$$n! = n(n-1)! \text{ pro } n > 1$$

```
int factorialI(int n) {
   int f = 1;
   for(; n > 1; --n) {
      f *= n;
   }
   return f;
}
```

```
int factorialR(int n) {
   int f = 1;
   if (n > 1) {
      f = n * factorialR(n-1);
   }
   return f;
}
```

lec05/DemoFactorial.java

Připomínka

Výpočet faktoriálu

- Iterace

$$n! = n(n-1)(n-2)\ldots 2\cdot 1$$

- Rekurze

$$n! = 1 \text{ pro } n \leq 1$$
$$n! = n(n-1)! \text{ pro } n > 1$$

```
int factorialI(int n) {
   int f = 1;
   for(; n > 1; --n) {
      f *= n;
   }
   return f;
}
```

```
int factorialR(int n) {
   int f = 1;
   if (n > 1) {
      f = n * factorialR(n-1);
   }
   return f;
}
```

lec05/DemoFactorial.java

Připomínka

Výpočet faktoriálu

- Iracace

$$n! = n(n-1)(n-2)\ldots 2\cdot 1$$

```java
int factorialI(int n) {
   int f = 1;
   for(; n > 1; --n) {
      f *= n;
   }
   return f;
}
```

- Rekurze

$$n! = 1 \text{ pro } n \leq 1$$
$$n! = n(n-1)! \text{ pro } n > 1$$

```java
int factorialR(int n) {
   int f = 1;
   if (n > 1) {
      f = n * factorialR(n-1);
   }
   return f;
}
```

lec05/DemoFactorial.java

Připomínka

Obsah

Faktoriál

Obrácený výpis

Karel

Hanojské věže

Rekurze

Fibonacciho posloupnost

Příklad výpis posloupnosti 1/3

- Vytvořte program, který přečte posloupnost čísel a vypíše ji v opačném pořadí
- Rozklad problému:
  - Zavedeme abstraktní příkaz: „*obrať posloupnost*"
  - Příkaz rozložíme do tří kroků:
    1. Přečti číslo

       číslo uložíme pro pozdější „obrácený" výpis
    2. Pokud není detekován konec posloupnost „*obrať posloupnost*"

       pokračujeme ve čtení čísel
    3. Vypiš číslo

       vypíšeme uložené číslo

Příklad výpis posloupnosti 1/3

- Vytvořte program, který přečte posloupnost čísel a vypíše ji v opačném pořadí
- Rozklad problému:
  - Zavedeme abstraktní příkaz: „*obrať posloupnost*"
  - Příkaz rozložíme do tří kroků:
    1. Přečti číslo

       číslo uložíme pro pozdější „obrácený" výpis
    2. Pokud není detekován konec posloupnost „*obrať posloupnost*"

       pokračujeme ve čtení čísel
    3. Vypiš číslo

       vypíšeme uložené číslo

Příklad výpis posloupnosti 2/3

```java
void reverse() {
   int v = scan.nextInt();
   if (scan.hasNext()) {
      reverse();
   }
   System.out.printf("%3d ", v);
}

public void start() {
   System.err.println("Enter a sequence of numbers ( use ctr+D for the end of the the sequence");
   reverse();
   System.out.println(""); //end of line
}
```

lec06/DemoRevertSequence.java

Příklad výpis posloupnosti 3/3

- lec06/DemoRevertSequence.java
- Vytvoření posloupností

```
./generate_numbers.sh | tr '\n' ' ' | cat > numbers.txt
javac DemoRevertSequence.java
java DemoRevertSequence < numbers.txt 2>/dev/null > numbers-
    r.txt
java DemoRevertSequence <numbers-r.txt 2>/dev/null > numbers
    -rr.txt
```

- Příkaz pro výpis obsahu souborů

```
for i in numbers.txt numbers-r.txt numbers-rr.txt;
    do echo "$i"; cat $i; echo ""; done
```

- Výpis obsahu souborů

```
numbers.txt
10 4 20 8 8 5 18 6 7 7
numbers-r.txt
  7   7   6  18   5   8   8  20   4  10

numbers-rr.txt
 10   4  20   8   8   5  18   6   7   7
```

Příklad výpis posloupnosti 3/3

- lec06/DemoRevertSequence.java
- Vytvoření posloupností

```
./generate_numbers.sh | tr '\n' ' ' | cat > numbers.txt
javac DemoRevertSequence.java
java DemoRevertSequence < numbers.txt 2>/dev/null > numbers-
    r.txt
java DemoRevertSequence <numbers-r.txt 2>/dev/null > numbers
    -rr.txt
```

- Příkaz pro výpis obsahu souborů

```
for i in numbers.txt numbers-r.txt numbers-rr.txt;
    do echo "$i"; cat $i; echo ""; done
```

- Výpis obsahu souborů

```
numbers.txt
10 4 20 8 8 5 18 6 7 7
numbers-r.txt
  7   7   6  18   5   8   8  20   4  10

numbers-rr.txt
 10   4  20   8   8   5  18   6   7   7
```

Příklad výpis posloupnosti 3/3

- lec06/DemoRevertSequence.java
- Vytvoření posloupností

```
./generate_numbers.sh | tr '\n' ' ' | cat > numbers.txt
javac DemoRevertSequence.java
java DemoRevertSequence < numbers.txt 2>/dev/null > numbers-
    r.txt
java DemoRevertSequence <numbers-r.txt 2>/dev/null > numbers
    -rr.txt
```

- Příkaz pro výpis obsahu souborů

```
for i in numbers.txt numbers-r.txt numbers-rr.txt;
    do echo "$i"; cat $i; echo ""; done
```

- Výpis obsahu souborů

```
numbers.txt
10 4 20 8 8 5 18 6 7 7
numbers-r.txt
  7   7   6  18   5   8   8  20   4  10

numbers-rr.txt
 10   4  20   8   8   5  18   6   7   7
```

Obsah

Faktoriál

Obrácený výpis

Karel

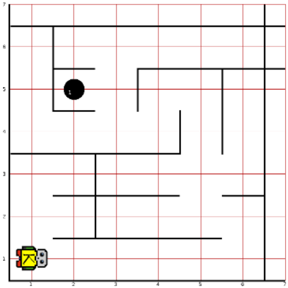
Hanojské věže

Rekurze

Fibonacciho posloupnost

Robot Karel – hledání „beeperu"

- Robot Karel má za úkol najít „beeper" v bludišti a vrátit se s ním zpět na výchozí pozici
- Karel se pohybuje v mřížkovém ortogonálním světě
  - Karel umí:
    - jít rovně o jeden krok
    - otočit se doleva o 90°
    - vzít / položit „beeper"
    - vypnout se
  - Karel umí zjistit, zda-li:
    - je před ním zeď
    - je na políčku, na kterém stojí, „beeper"



Robot Karel – Iterační řešení 1/3

- Návrh řešení
  1. Jdi podél zdi (po pravé straně) dokud není na políčku „beeper" a počítej počet kroků;
  2. Seber „beeper";
  3. Otoč se a jdi podél zdi (po levé straně) a udělej stejný počet kroků jako při hledání „beeperu";
  4. Polož „beeper" a vypni se.
- Abstraktní příkazy pro nalezení „beeperu" sledování zdi na pravé straně:
  - Otoč se doprava: **turnRight**
  - Zjisti, zda-li je na pravé straně zeď: **isWallOnRight**
  - Jdi podél pravé zdi: **goByRightWall**
- Analogické abstraktní příkazy pro pohyb podél zdi na levé straně:
  - **isWallOnLeft** a **goByLeftWall**

Robot Karel – Iterační řešení 1/3

- Návrh řešení
  1. Jdi podél zdi (po pravé straně) dokud není na políčku „beeper" a počítej počet kroků;
  2. Seber „beeper";
  3. Otoč se a jdi podél zdi (po levé straně) a udělej stejný počet kroků jako při hledání „beeperu";
  4. Polož „beeper" a vypni se.
- Abstraktní příkazy pro nalezení „beeperu" sledování zdi na pravé straně:
  - Otoč se doprava: **turnRight**
  - Zjisti, zda-li je na pravé straně zeď: **isWallOnRight**
  - Jdi podél pravé zdi: **goByRightWall**
- Analogické abstraktní příkazy pro pohyb podél zdi na levé straně:
  - **isWallOnLeft** a **goByLeftWall**

Robot Karel – Iterační řešení 1/3

- Návrh řešení
  1. Jdi podél zdi (po pravé straně) dokud není na políčku „beeper" a počítej počet kroků;
  2. Seber „beeper";
  3. Otoč se a jdi podél zdi (po levé straně) a udělej stejný počet kroků jako při hledání „beeperu";
  4. Polož „beeper" a vypni se.
- Abstraktní příkazy pro nalezení „beeperu" sledování zdi na pravé straně:
  - Otoč se doprava: **turnRight**
  - Zjisti, zda-li je na pravé straně zeď: **isWallOnRight**
  - Jdi podél pravé zdi: **goByRightWall**
- Analogické abstraktní příkazy pro pohyb podél zdi na levé straně:
  - **isWallOnLeft** a **goByLeftWall**

Robot Karel – Iterační řešení 2/3

```
void turnRight() {
   for (int i = 0; i < 3; i++) {
      turnLeft();
   }
}

boolean isWallOnRight() {
   turnRight();
   boolean out = isInFrontOfWall();
   turnLeft();
   return out;
}

void goByRightWall() {
   while (!isOnBeeper()) {
      if (!wallOnRight()) {
         turnRight();
      }
      while (isInFrontOfWall()) {
         turnLeft();
      }
      step();
   }
}
```

Analogicky pro zeď po levé straně

Robot Karel – Iterační řešení 3/3

- Počet kroků je inkrementován v proceduře **step**
- Při návratu podél levé zdi Karel ujde stejný počet kroků

```
public void execute() {
   goByRight();
   pickBeeper();
   turnAround();

   if (stepsTaken > 0) {
      goByLeft(stepsTaken);
   }

   putBeeper();
   turnOff();
}
```

Robot Karel – Rekurzivní řešení 1/3

- Řešení rekurzí založíme na opakovaném volání funkce, která posune robot pouze o jeden krok vpřed a následně jej vrátí zpět ve funkci **go**:
  1. Pokud jsi na „beeperu" seber jej a otoč o 180° a ukonči hledání
  2. Zapamatuj si svou orientaci a udělej jeden krok podél pravé zdi
  3. **go** (opakuj hledání o jeden krok)
  4. Udělej jeden krok zpět (vrať se o jeden krok zpět a natoč se do původního směru)

Robot Karel – Rekurzivní řešení 2/3

```
void go() {
   if (isOnBeeper()) {
      pickBeeper();
      turnAround(); //turn back
      return;
   }
   int numberTurnRight = 0;
   if (!isWallOnRight()) {
      turnRight();
      numberTurnRight++;
   }

   while (isInFrontOfWall()) {
      turnLeft();
      numberTurnRight--;
   }
   move(); //move forward
   go();
   move(); //move backward

   numberTurnRight = (numberTurnRight + 4) % 4;
   for (int i = 0; i < numberTurnRight; ++i) {
      turnLeft(); //revert the turns
   }
}
```

Robot Karel – Rekurzivní řešení 3/3

- Hlavní procedura obsahuje pouze volání funkce **go**, která vrací robot na výchozí políčko

```
void execute() {
   go();
   putBeeper();
   turnOff();
}
```

- Pro spuštění využijeme knihovnu `KarelJRobot.jar` a modifikovaný svět `maze.klwd`

```
javac -cp lib/KarelJRobot.jar DemoKarel.java
java -cp lib/KarelJRobot.jar:. DemoKarel
```

- Robot se při zpáteční cestě vrací „rychleji", protože již explicitně nekontroluje, zda-li je zeď na levé straně.
- Uvedený demonstrační program není plným řešení úlohy ze 3. cvičení

`lec06/DemoKarel.java`

Obsah

Faktoriál

Obrácený výpis

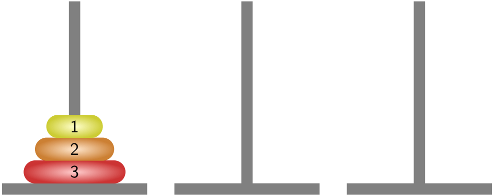
Karel

Hanojské věže

Rekurze

Fibonacciho posloupnost

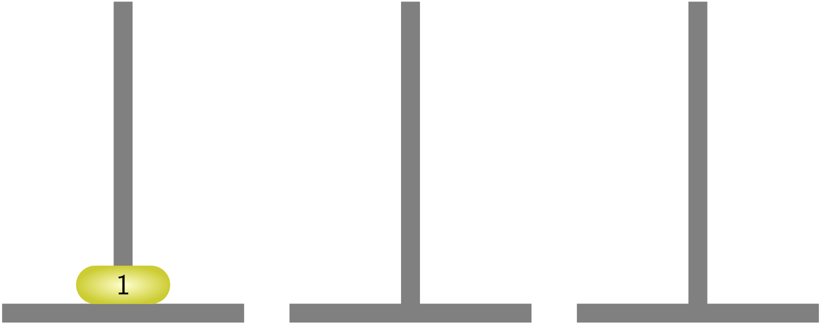
Příklad Hanojské věže



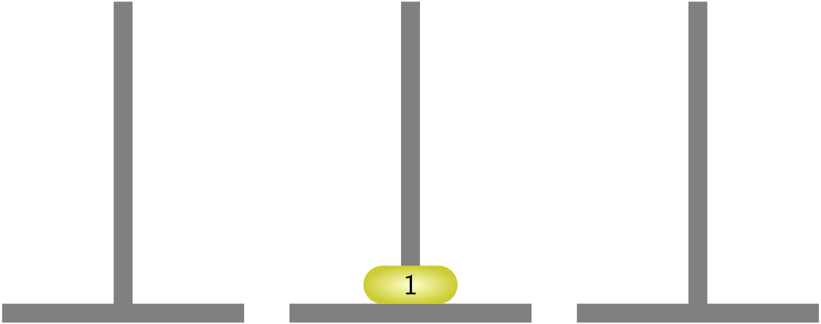
- Přemístit disky na druhou jehlu s použitím třetí (pomocné) jehly za dodržení pravidel:
  1. V každém kroku můžeme přemístit pouze jeden disk a to vždy z jehly na jehlu

     Disky nelze odkládat mimo jehly
  2. Položit větší disk na menší není dovoleno

Hanojské věže – 1 disk

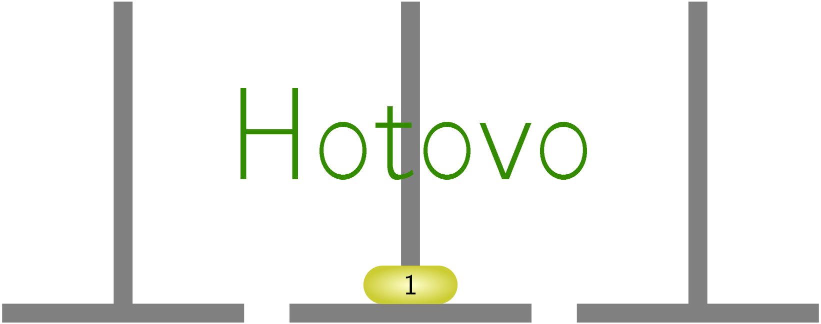


Hanojské věže – 1 disk

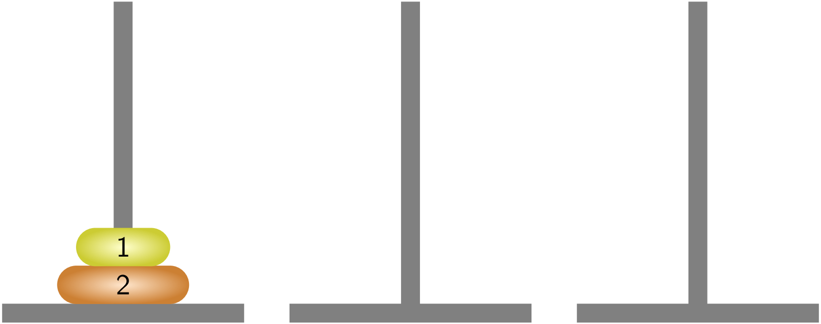


Přesunutý disk z jehly 1 na jehlu 2.

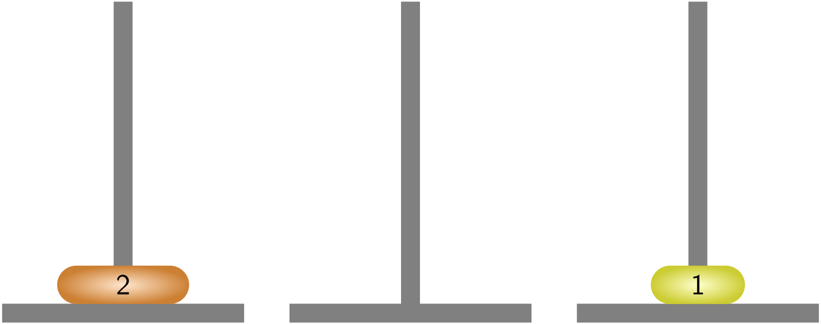
Hanojské věže – 1 disk



Hanojské věže – 2 disky

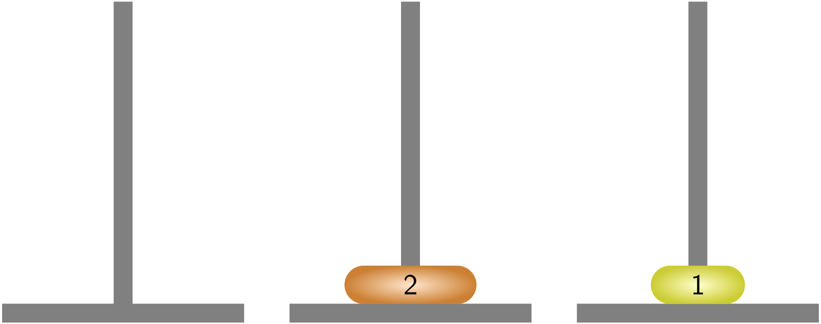


Hanojské věže – 2 disky



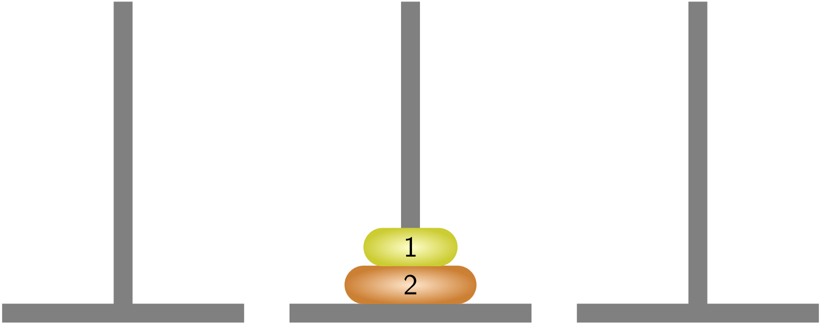
Přesunutý disk z jehly 1 na jehlu 3.

Hanojské věže – 2 disky



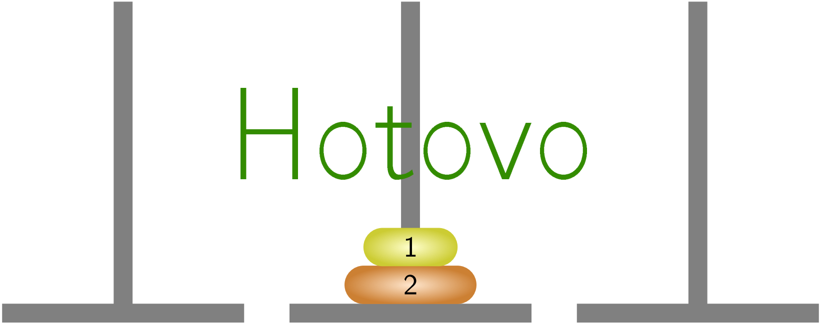
Přesunutý disk z jehly 1 na jehlu 2.

Hanojské věže – 2 disky

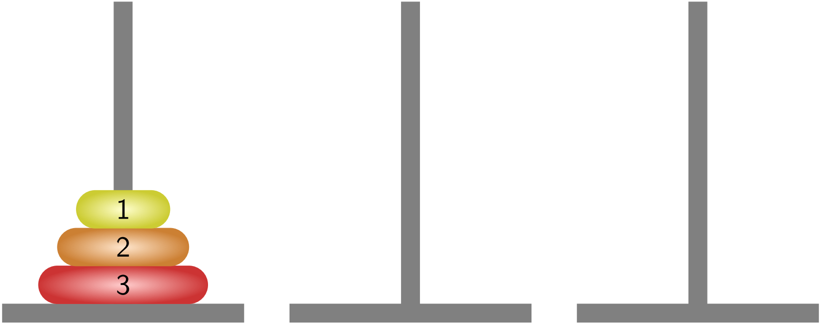


Přesunutý disk z jehly 3 na jehlu 2.

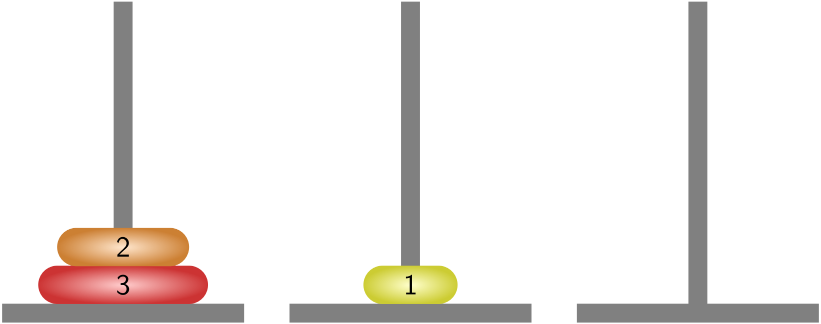
Hanojské věže – 2 disky



Hanojské věže – 3 disky

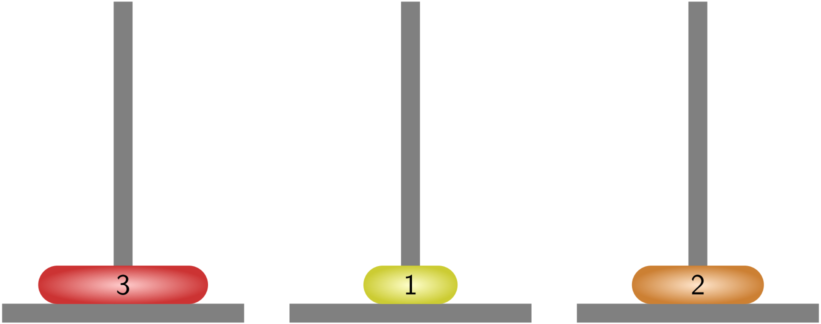


Hanojské věže – 3 disky



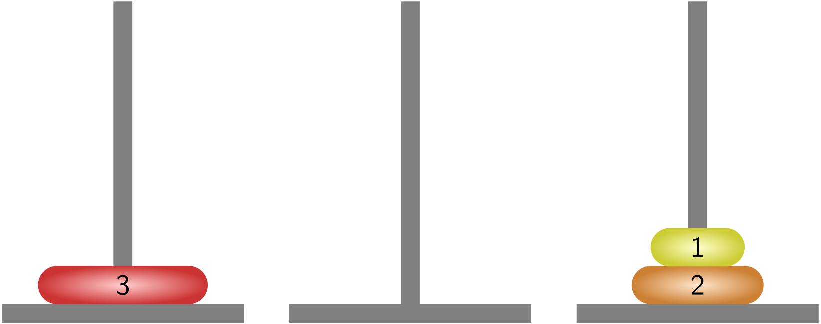
Přesunutý disk z jehly 1 na jehlu 2.

Hanojské věže – 3 disky



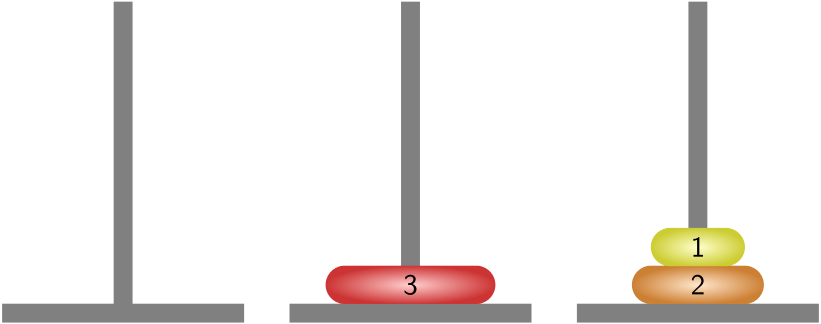
Přesunutý disk z jehly 1 na jehlu 3.

Hanojské věže – 3 disky



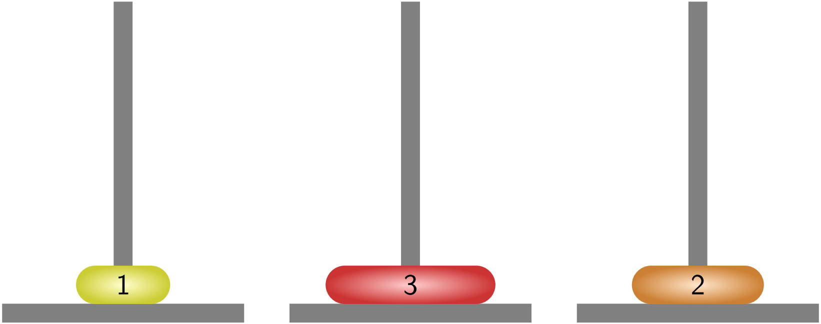
Přesunutý disk z jehly 2 na jehlu 3.

Hanojské věže – 3 disky



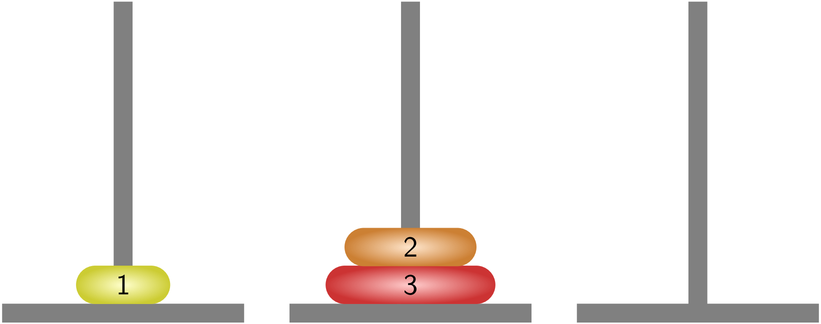
Přesunutý disk z jehly 1 na jehlu 2.

Hanojské věže – 3 disky



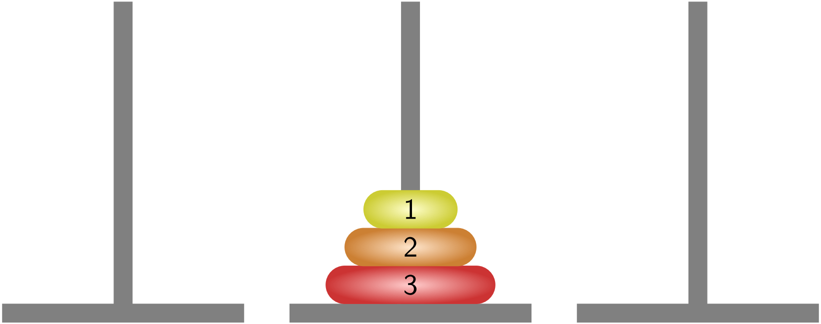
Přesunutý disk z jehly 3 na jehlu 1.

Hanojské věže – 3 disky



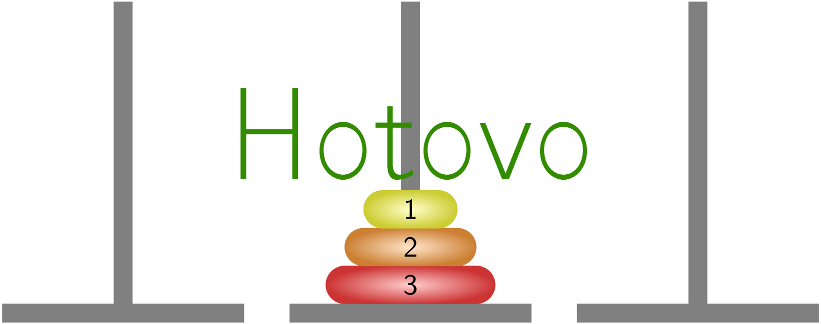
Přesunutý disk z jehly 3 na jehlu 2.

Hanojské věže – 3 disky

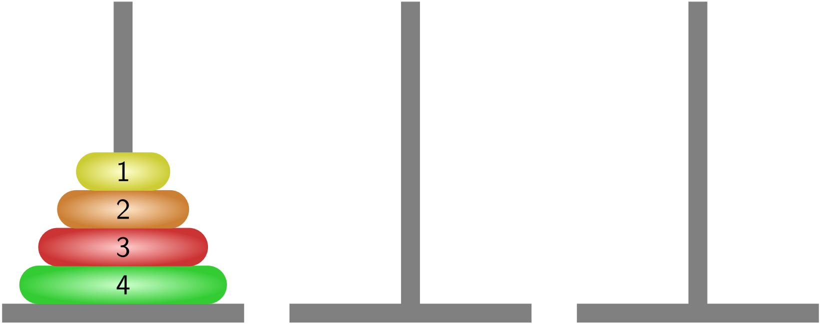


Přesunutý disk z jehly 1 na jehlu 2.

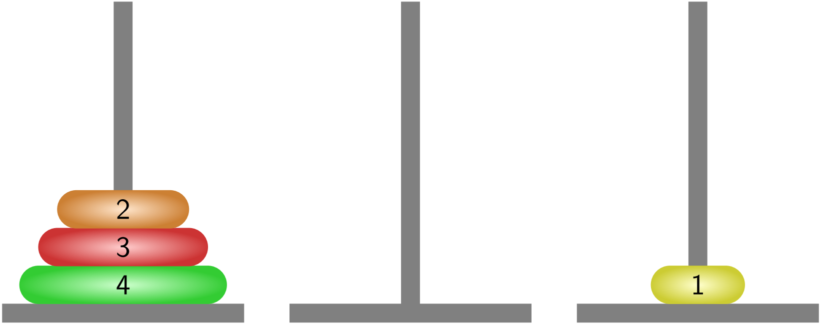
Hanojské věže – 3 disky



Hanojské věže – 4 disky

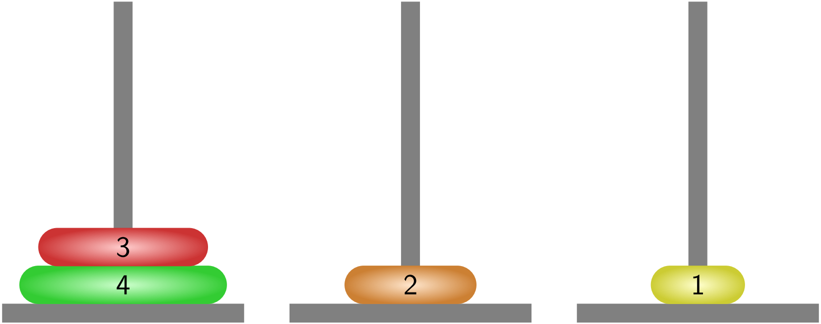


Hanojské věže – 4 disky



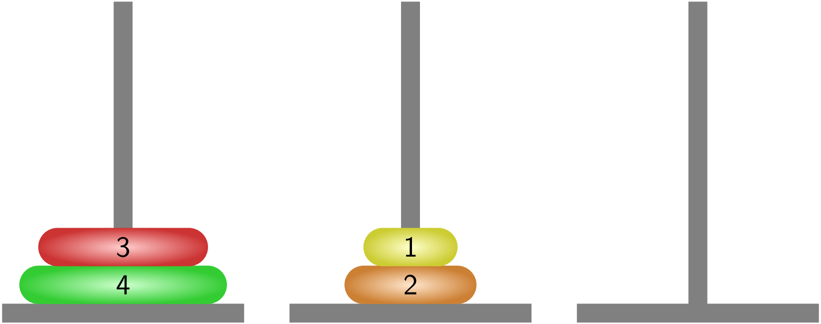
Přesunutý disk z jehly 1 na jehlu 3.

Hanojské věže – 4 disky



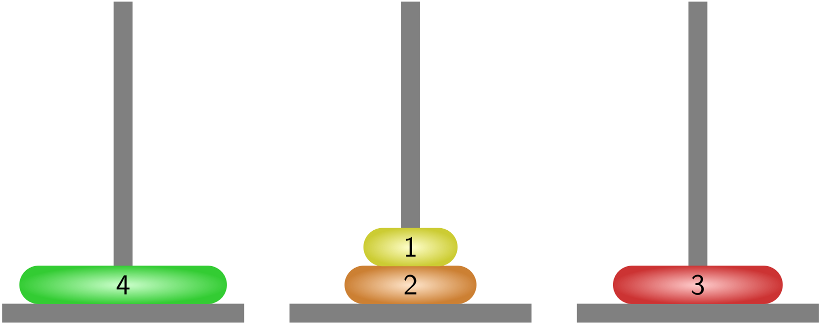
Přesunutý disk z jehly 1 na jehlu 2.

Hanojské věže – 4 disky



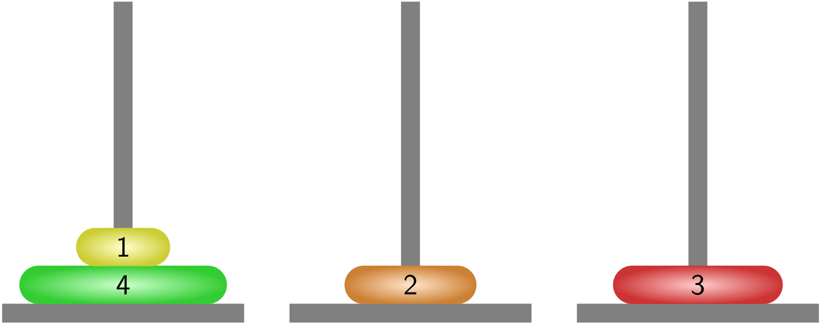
Přesunutý disk z jehly 3 na jehlu 2.

Hanojské věže – 4 disky



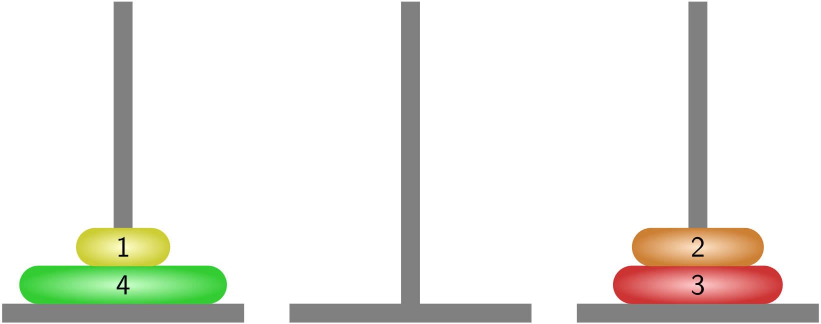
Přesunutý disk z jehly 1 na jehlu 3.

Hanojské věže – 4 disky



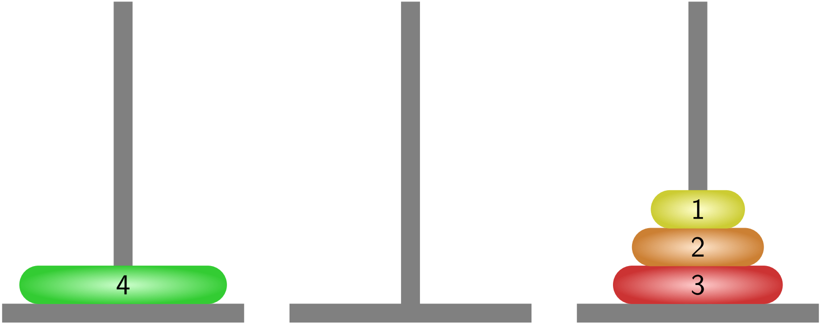
Přesunutý disk z jehly 2 na jehlu 1.

Hanojské věže – 4 disky



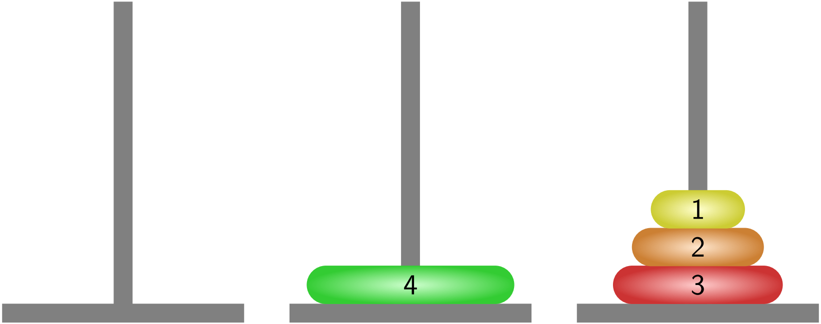
Přesunutý disk z jehly 2 na jehlu 3.

Hanojské věže – 4 disky



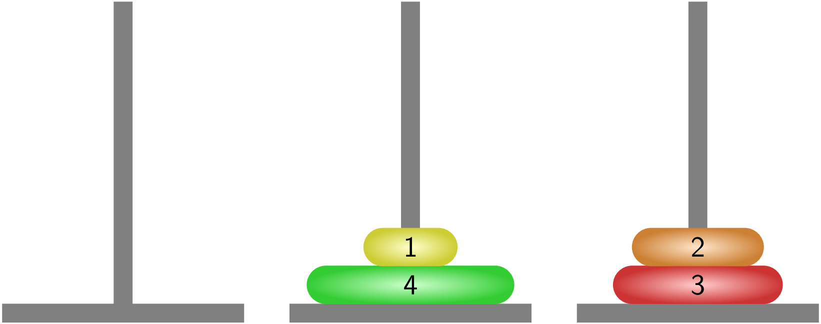
Přesunutý disk z jehly 1 na jehlu 3.

Hanojské věže – 4 disky



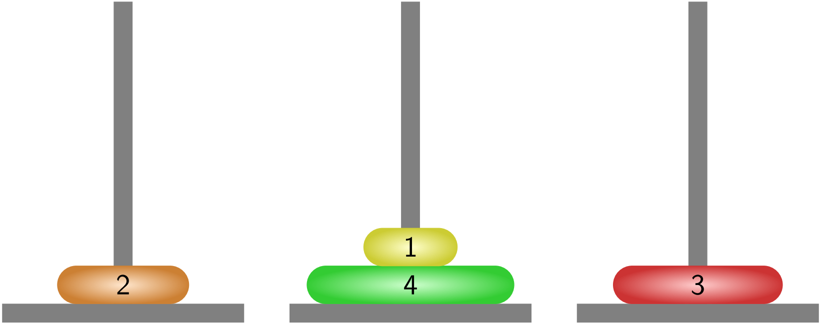
Přesunutý disk z jehly 1 na jehlu 2.

Hanojské věže – 4 disky



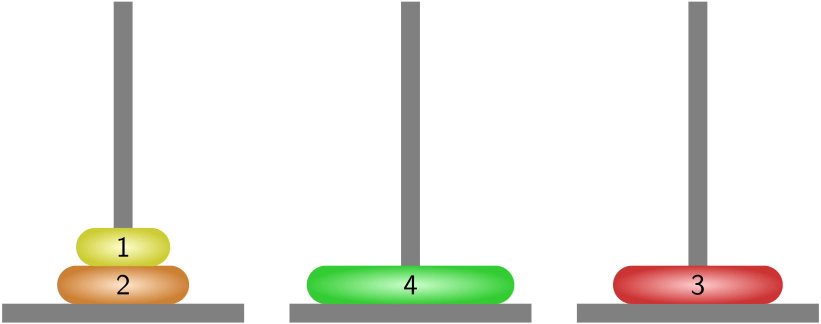
Přesunutý disk z jehly 3 na jehlu 2.

Hanojské věže – 4 disky



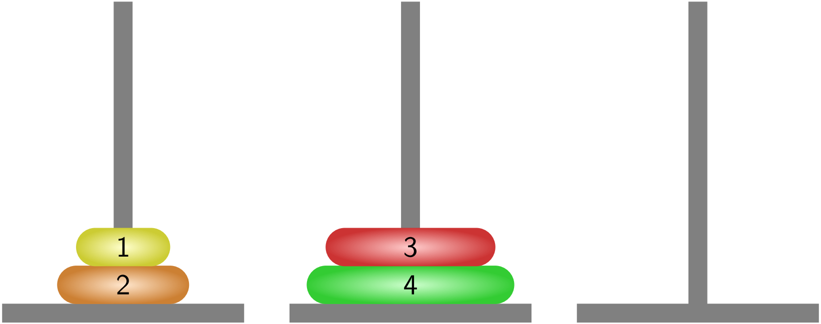
Přesunutý disk z jehly 3 na jehlu 1.

Hanojské věže – 4 disky



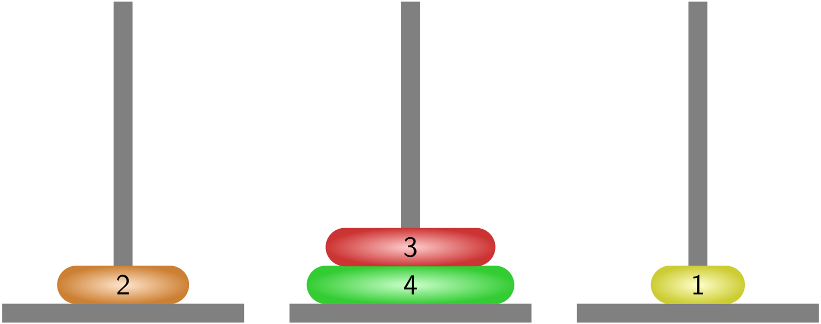
Přesunutý disk z jehly 2 na jehlu 1.

Hanojské věže – 4 disky



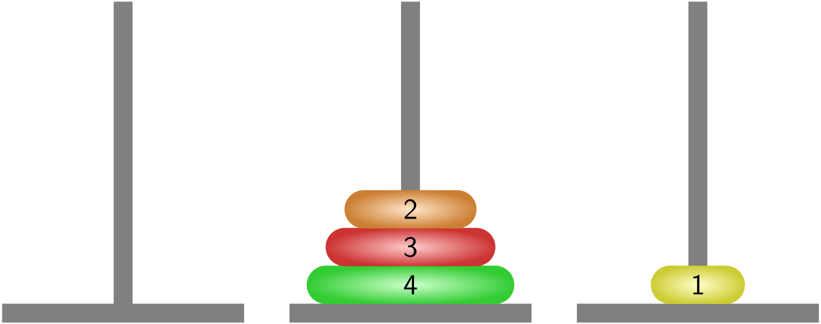
Přesunutý disk z jehly 3 na jehlu 2.

Hanojské věže – 4 disky



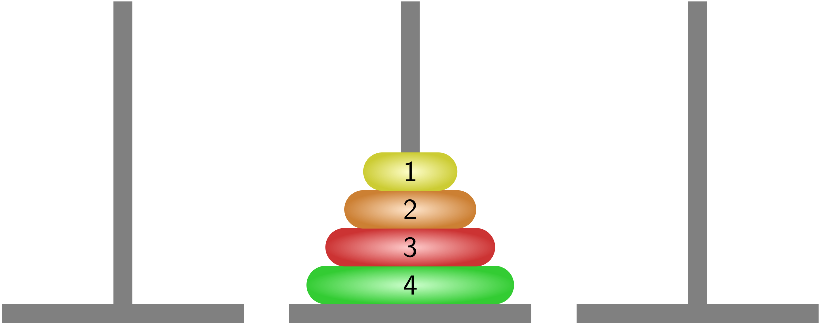
Přesunutý disk z jehly 1 na jehlu 3.

Hanojské věže – 4 disky



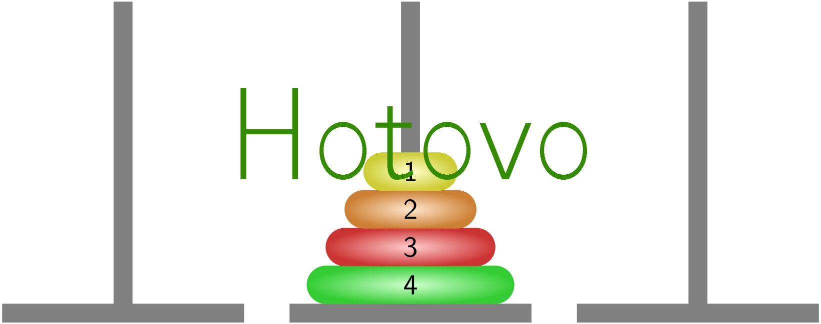
Přesunutý disk z jehly 1 na jehlu 2.

Hanojské věže – 4 disky

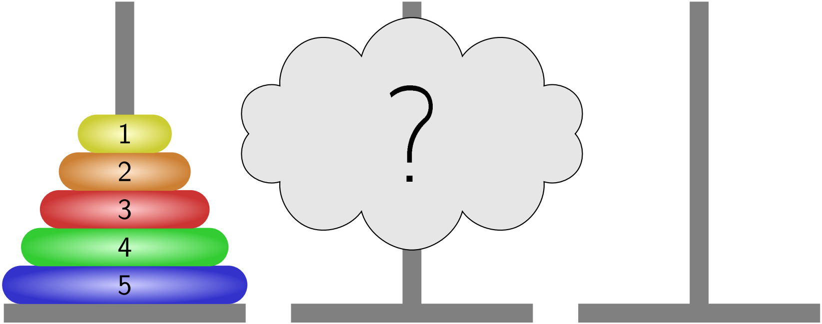


Přesunutý disk z jehly 3 na jehlu 2.

Hanojské věže – 4 disky



Hanojské věže – 5 disků



Návrh řešení

- Zavedeme abstraktní příkaz **moveTower(n, 1, 2, 3)** realizující přesun n disků z jehly 1 na jehlu 2 s použitím jehly 3.
- Pro $n > 0$ můžeme příkaz rozložit na tři jednodušší příkazy
  1. **moveTower(n-1, 1, 3, 2)**
     přesun $n-1$ disků z jehly 1 na jehlu 3
  2. „*přenes disk z jehly na jehlu 2*"
     přesun největšího disku na cílovou pozici
     abstraktní příkaz
  3. **moveTower(n-1, 3, 2, 1)**
     přesun $n-1$ disků na cílovou pozici

Návrh řešení

- Zavedeme abstraktní příkaz **moveTower(n, 1, 2, 3)** realizující přesun n disků z jehly 1 na jehlu 2 s použitím jehly 3.
- Pro $n > 0$ můžeme příkaz rozložit na tři jednodušší příkazy
  1. **moveTower(n-1, 1, 3, 2)**
     přesun $n - 1$ disků z jehly 1 na jehlu 3
  2. „*přenes disk z jehly na jehlu 2*"
     přesun největšího disku na cílovou pozici
     abstraktní příkaz
  3. **moveTower(n-1, 3, 2, 1)**
     přesun $n - 1$ disků na cílovou pozici

Příklad řešení

```java
void moveTower(int n, int from, int to, int tmp) {
   if (n > 0) {
      moveTower(n - 1, from, tmp, to); //move to tmp
      System.out.println("Move disc from " + from + "
 to " + to);
      moveTower(n - 1, tmp, to, from); //move from tmp
   }
}

void start() {
   int numberOfDiscs = 5;
   moveTower(numberOfDiscs, 1, 2, 3);
}
```

lec06/DemoTowersOfHanoi.java

Příklad výpisu

- lec06/DemoTowersOfHanoi.java

```
java DemoTowersOfHanoi 3
Move disc from 1 to 2
Move disc from 1 to 3
Move disc from 2 to 3
Move disc from 1 to 2
Move disc from 3 to 1
Move disc from 3 to 2
Move disc from 1 to 2
```

```
java DemoTowersOfHanoi 4
Move disc from 1 to 3
Move disc from 1 to 2
Move disc from 3 to 2
Move disc from 1 to 3
Move disc from 2 to 1
Move disc from 2 to 3
Move disc from 1 to 3
Move disc from 1 to 2
Move disc from 3 to 2
Move disc from 3 to 1
Move disc from 2 to 1
Move disc from 3 to 2
Move disc from 1 to 3
Move disc from 1 to 2
Move disc from 3 to 2
```

Obsah

Faktoriál

Obrácený výpis

Karel

Hanojské věže

Rekurze

Fibonacciho posloupnost

Rekurzivní algoritmy

- Rekurzivní funkce jsou přímou realizací rekurzivních algoritmů
- Rekurzivní algoritmus předepisuje výpočet „**shora dolů**" v závislosti na velikosti vstupních dat
  - Pro nejmenší (nejjednodušší) vstup je výpočet předepsán přímo
  - Pro obecný vstup je výpočet předepsán s využitím téhož algoritmu pro menší vstup
- Výhodou rekurzivních funkcí je jednoduchost a přehlednost

Rekurzivní vs iteračními algoritmy

- Nevýhodou rekurzivních algoritmů může být časová náročnost způsobená např. zbytečným opakováním výpočtu
- Řadu rekurzivních algoritmů lze nahradit iteračními, které počítají výsledek „**zdola nahoru**, tj. od menších (jednodušších) vstupních dat k větším (složitějším).
- Pokud algoritmus výpočtu „**zdola nahoru**" nenajdeme, např. při řešení problému Hanojských věží, lze rekurzivitu odstranit pomocí zásobníku.

Např. zásobník využijeme pro uložení stavu řešení problému.

Rekurze

"To iterate is human, to recurse divine."

L. Peter Deutsch

http://www.devtopics.com/101-great-computer-programming-quotes

Elegance vs obtížnost rekurze

I've often heard people describe understanding recursion as one of those "got it" moments, when the universe opened its secret stores of knowledge and gifted the mind of a burgeoning developer with a very powerful tool. For me, recursion has always been hard. Each time I'm able to peer more into its murky depths, I am humbled to see how little I feel like I really appreciate and understand its power and elegance.

Rick Winfrey, 2012

`http://selfless-singleton.rickwinfrey.com/2012/11/27/to-iterate-is-human-to-recurse-divine`

Obsah

Faktoriál

Obrácený výpis

Karel

Hanojské věže

Rekurze

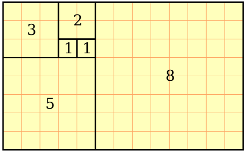
Fibonacciho posloupnost

Fibonacciho posloupnost

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

  Nebo 0, 1, 1, 2, 3, 5, ...
- $F_n = F_{n-1} + F_{n-2}$
- pro $F_1 = 1,\ F_2 = 1$

  Nebo $F_1 = 0, F_2 = 1$



- Nekonečná posloupnost přirozených čísel, kde každé číslo je součtem dvou předchozích.
- Limita poměru dvou následujících čísel Fibonacciho posloupnosti je rovna zlatému řezu.
  - Sectio aurea – ideální poměr mezi různými délkami
  - Rozdělení úsečky na dvě části tak, že poměr větší části ku menší je stejný jako poměr celé úsečky k větší části $\varphi = \frac{1+\sqrt{5}}{2} \approx 1{,}618\,033\,988\,749\,894\,848\ldots$

Fibonacciho posloupnost – historie

- Indičtí matematici (450 nebo 200 BC)
- Leonardo Pisano (1175–1250) popis růstu populace králíků

  italský matematik známý také jako Fibonacci

  - F_n – velikost populace po n měsících za předpokladu
    - První měsíc se narodí jediný pár.
    - Narozené páry jsou produktivní od 2. měsíce svého života.
    - Každý měsíc zplodí každý produktivní pár jeden další pár.
    - Králíci nikdy neumírají, nejsou nemocní atd.
- Henry E. Dudeney (1857–1930) – popis populace krav
  - „Jestliže každá kráva vyprodukuje své první tele (jalovici) za rok a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže žádná nezemře a na počátku budete mít jednu krávu?

    Po 12 let je k dispozici jeden či více býků

Fibonacciho posloupnost – historie

- Indičtí matematici (450 nebo 200 BC)
- Leonardo Pisano (1175–1250) popis růstu populace králíků

  italský matematik známý také jako Fibonacci

  - F_n – velikost populace po n měsících za předpokladu
    - První měsíc se narodí jediný pár.
    - Narozené páry jsou produktivní od 2. měsíce svého života.
    - Každý měsíc zplodí každý produktivní pár jeden další pár.
    - Králíci nikdy neumírají, nejsou nemocní atd.
- Henry E. Dudeney (1857–1930) – popis populace krav
  - „Jestliže každá kráva vyprodukuje své první tele (jalovici) za rok a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže žádná nezemře a na počátku budete mít jednu krávu?

    Po 12 let je k dispozici jeden či více býků

Fibonacciho posloupnost – rekurzivně

- Platí:

$$f_0 = 1$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}, \text{ pro } n > 1$$

```c
int fibonacci(int n) {
   return n < 2
      ? 1
      : fibonacci(n - 1) + fibonacci(n - 2);
}
```

Zápis je elegantní, jak je však takový výpočet efektivní?

Fibonacciho posloupnost – rekurzivně

- Platí:

$$f_0 = 1$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}, \text{ pro } n > 1$$

```
int fibonacci(int n) {
   return n < 2
      ? 1
      : fibonacci(n - 1) + fibonacci(n - 2);
}
```

Zápis je elegantní, jak je však takový výpočet efektivní?

Fibonacciho posloupnost – příklad 1/2

- Počet operací při výpočtu Fibonacciho čísla n

```
long counter;

long fibonnaciR(int n) {
   counter++;
   return n < 2 ? 1 : fibonnaciR(n-1) + fibonnaciR(n-2);
}

long fibonnaciI(int n) {
   long fibM2 = 1l;
   long fibM1 = 1l;
   long fib = 1l;
   for (int i = 2; i <= n; ++i) {
      fibM2 = fibM1;
      fibM1 = fib;
      fib = fibM1 + fibM2;
      counter += 3;
   }
   return fib;
}
```

lec06/DemoFibonacci.java

Fibonacciho posloupnost – rekurzivně 2/2

```java
public void start(String[] args) {
   int n = args.length > 0 ? Integer.parseInt(args[0]) : 25;
   counter = 0;
   long fibR = fibonnaciR(n);
   long counteR = counter;

   counter = 0;
   long fibI = fibonnaciI(n);
   long counteI = counter;

   System.out.println("Fibonacci number recursive: " + fibR);
   System.out.println("Fibonacci number iteration: " + fibI);
   System.out.println("Counter recursive: " + counteR);
   System.out.println("Counter recursive: " + counteI);
}
```

lec06/DemoFibonacci.java

```
javac DemoFibonacci.java
java DemoFibonacci 30
```

```
Fibonacci number recursive: 1346269
Fibonacci number iteration: 1346269
Counter recursive: 2692537
Counter iteration: 8
```

Fibonacciho posloupnost – rekurzivně vs iteračně

- Rekurzivní výpočet
  - Složitost roste exponenciálně s $n \sim 2^n$
- Iterační algoritmus
  - Počet operací je proporcionální $n \sim 3n$
- Skutečný počet operací závisí na konkrétní implementaci, programovacím jazyku, překladači a hardware
- Složitost algoritmů proto vyjadřujeme asymptoticky jako funkci velikosti vstupu
  - Například v tzv. „Big O" notaci
    - rekurzivní algoritmus výpočtu má složitost $O(2^n)$
    - iterační algoritmus výpočtu má složitost $O(n)$

Efektivní algoritmy mají polynomiální složitost

Fibonacciho posloupnost – rekurzivně vs iteračně

- Rekurzivní výpočet
  - Složitost roste exponenciálně s $n \sim 2^n$
- Iterační algoritmus
  - Počet operací je proporcionální $n \sim 3n$
- Skutečný počet operací závisí na konkrétní implementaci, programovacím jazyku, překladači a hardware
- Složitost algoritmů proto vyjadřujeme asymptoticky jako funkci velikosti vstupu
  - Například v tzv. „Big O" notaci
    - rekurzivní algoritmus výpočtu má složitost $O(2^n)$
    - iterační algoritmus výpočtu má složitost $O(n)$

Efektivní algoritmy mají polynomiální složitost

Část II

Příklady

Obsah

Eratosthenovo síto

Rozklad na prvočinitele

Řazení

Pole reprezentující množinu prvočísel

- Vypsat všechna prvočísla menší nebo rovna zadané hodnotě `max`
- Algoritmus:
  1. Vytvoříme množinu obsahující všechna přirozená čísla od 2 do `max`
  2. Z množiny vypustíme násobky čísla 2
  3. Najdeme nejbližší číslo k tomu, jehož násobky jsme v předchozím kroku vypustili, a vypustíme všechny násobky tohoto čísla
  4. Opakujeme krok 3, dokud číslo, jehož násobky jsme vypustili, není větší než odmocnina z `max`
  5. Čísla, která v množině zůstanou, jsou hledaná prvočísla
- Pro reprezentaci množiny čísel použijeme pole prvků typu `boolean`, kde prvek pole `sieve[i]` udává, zda-li je celé číslo i v množině (`true`) nebo není (`false`), tj. zda-li je nebo není prvočíslem

Pole reprezentující množinu prvočísel

- Vypsat všechna prvočísla menší nebo rovna zadané hodnotě `max`
- Algoritmus:
  1. Vytvoříme množinu obsahující všechna přirozená čísla od 2 do `max`
  2. Z množiny vypustíme násobky čísla 2
  3. Najdeme nejbližší číslo k tomu, jehož násobky jsme v předchozím kroku vypustili, a vypustíme všechny násobky tohoto čísla
  4. Opakujeme krok 3, dokud číslo, jehož násobky jsme vypustili, není větší než odmocnina z `max`
  5. Čísla, která v množině zůstanou, jsou hledaná prvočísla
- Pro reprezentaci množiny čísel použijeme pole prvků typu `boolean`, kde prvek pole `sieve[i]` udává, zda-li je celé číslo *i* v množině (`true`) nebo není (`false`), tj. zda-li je nebo není prvočíslem

Pole reprezentující množinu prvočísel

- Vypsat všechna prvočísla menší nebo rovna zadané hodnotě `max`
- Algoritmus:
  1. Vytvoříme množinu obsahující všechna přirozená čísla od 2 do `max`
  2. Z množiny vypustíme násobky čísla 2
  3. Najdeme nejbližší číslo k tomu, jehož násobky jsme v předchozím kroku vypustili, a vypustíme všechny násobky tohoto čísla
  4. Opakujeme krok 3, dokud číslo, jehož násobky jsme vypustili, není větší než odmocnina z `max`
  5. Čísla, která v množině zůstanou, jsou hledaná prvočísla
- Pro reprezentaci množiny čísel použijeme pole prvků typu `boolean`, kde prvek pole `sieve[i]` udává, zda-li je celé číslo *i* v množině (`true`) nebo není (`false`), tj. zda-li je nebo není prvočíslem

Eratosthenovo síto 1/2

```java
boolean[] createSieve(int max) {
   boolean[] sieve = new boolean[max + 1];
   for (int i = 2; i <= max; ++i) {
      sieve[i] = true;
   }
   int p = 2;
   int pmax = (int)Math.sqrt(max);
   do {
      for(int i = p + p; i <= max; i += p) {
         sieve[i] = false;
      }
      do {
         p++;
      } while (!sieve[p]);
   } while (p <= pmax);
   return sieve;
}
```

lec06/DemoSieveOfEratosthenes.java

Eratosthenovo síto 2/2

```
void print(boolean[] sieve) {
   for(int i = 2; i < sieve.length; ++i) {
      if (sieve[i]) {
         System.out.print(i + " ");
      }
   }
}

void start(int max) {
   boolean[] sieve = createSieve(max);
   System.out.print("Prime numbers from 2 to " + max + ": ");
   print(sieve);
   System.out.println("");
}
```

```
javac DemoSieveOfEratosthenes.java
java DemoSieveOfEratosthenes
Prime numbers from 2 to 100: 2 3 5 7 11 13 17 19 23 29 31
    37 41 43 47 53 59 61 67 71 73 79 83 89 97
java DemoSieveOfEratosthenes 30
Prime numbers from 2 to 40: 2 3 5 7 11 13 17 19 23 29
```

lec06/DemoSieveOfEratosthenes.java

Obsah

Eratosthenovo síto

Rozklad na prvočinitele

Řazení

Rozklad na prvočinitele 1/2

- Rozklad přirozeného čísla n na součin prvočísel
- Řešení:
  - Dělit 2, pak 3, pak 5 a dalšími prvočísly, až $n - 1$
  - Každé dělení beze zbytku dodá jednoho prvočinitele
- Příklad:
  - $60 / 2 \rightarrow 30 / 2 \rightarrow 15 / 3 \rightarrow 5 / 5$
  - 60 má prvočinitele 2, 2, 3, 5

Rozklad na prvočinitele 2/2

Iterační algoritmus

```
void getPrimesI(int x) {
   int d = 2;
   while (d < x) {
      while (d < x && x % d !=
    0) {
         d++;
      }
      System.out.print(d + " ");
      x = x / d;
   }
}
```

Rekurzivní algoritmus

```
void getPrimesR(int x, int d) {
   if (d < x) {
      while (d < x && x % d !=
    0) {
         d++;
      }
      System.out.print(d + " ");
      getPrimesR(x / d, d);
   }
}
```

```
  void start(int n) {
     System.out.println("Decomposition of "  + n + " to primies");
     System.out.print("Iterative algorithm: ");
     getPrimesI(n);
     System.out.println("");
     System.out.print("Recursive algorithm: ");
     getPrimesR(n, 2);
     System.out.println("");
  }
```

lec06/DemoPrimes.java

Obsah

Eratosthenovo síto

Rozklad na prvočinitele

Řazení

Řazení pole

- Problém seřadit množinu prvků podle klíče (celého čísla)
- Posloupnost prvků $q = \langle a_1, \ldots, a_n \rangle$
- Délka posloupnosti q je $|q| = n$
- Posloupnost q je seřazená, právě tehdy když
  - $|q| < 2$
  - $|q| \geq 2$ **klíč**$(a_1) \leq$ **klíč**(a_2) a posloupnost $\langle a_2, \ldots \rangle$ neobsahuje prvek a_1.
- Řazení polí je řazením na místě

Pro jednoduchost v příkladech třídíme jen celá čísla, prakticky však zpravidla třídíme klíče datových položek.

Řazení pole

- Problém seřadit množinu prvků podle klíče (celého čísla)
- Posloupnost prvků $q = \langle a_1, \ldots, a_n \rangle$
- Délka posloupnosti q je $|q| = n$
- Posloupnost q je seřazená, právě tehdy když
  - $|q| < 2$
  - $|q| \geq 2$ **klíč(a_1)** $\leq$ **klíč(a_2)** a posloupnost $\langle a_2, \ldots \rangle$ neobsahuje prvek a_1.
- Řazení polí je řazením na místě

Pro jednoduchost v příkladech třídíme jen celá čísla, prakticky však zpravidla třídíme klíče datových položek.

Třídění přímým vkládáním – Insert Sort 1/2

Příklad - Třídění přímým vkládáním

1		*2*		*3*		*4*		*5*		*6*
44		55		12		42		94		18

1

2

3

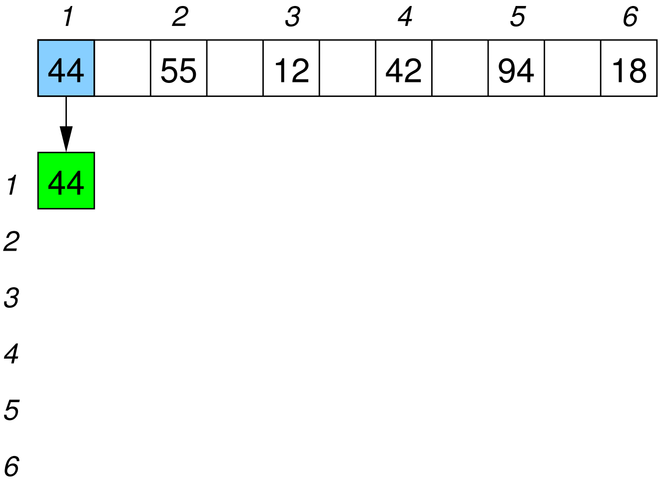
4

5

6

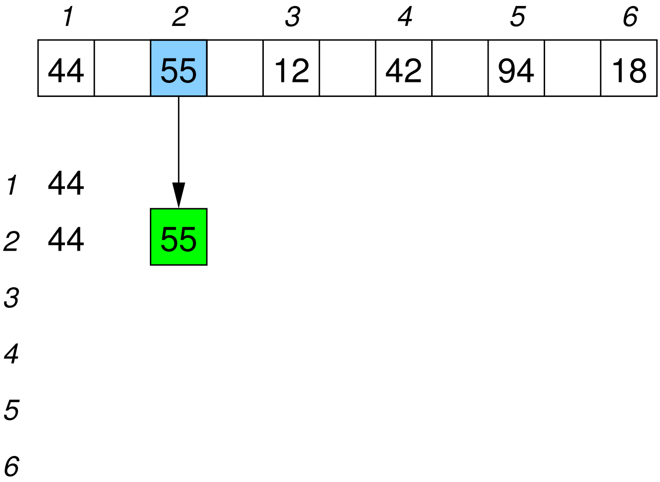
Třídění přímým vkládáním – Insert Sort 1/2

Příklad - Třídění přímým vkládáním



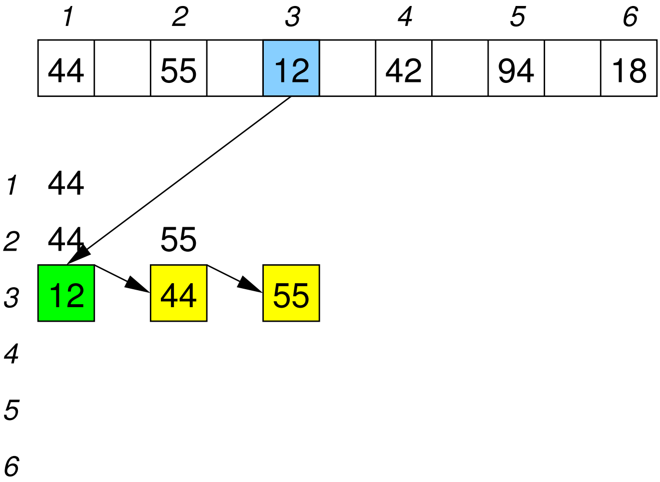
Třídění přímým vkládáním – Insert Sort 1/2

Příklad - Třídění přímým vkládáním



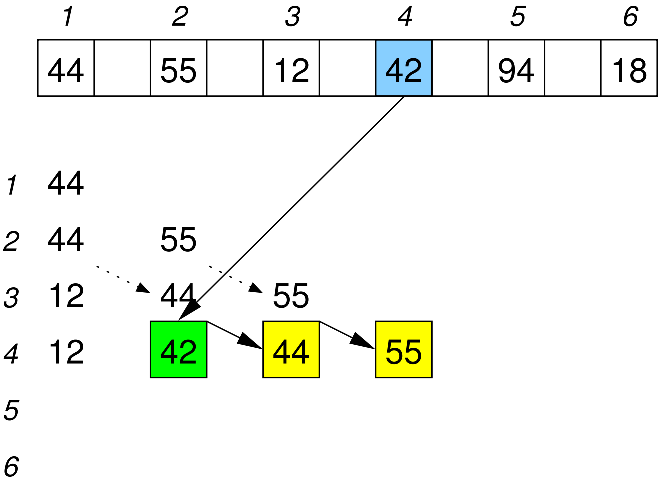
Třídění přímým vkládáním – Insert Sort 1/2

Příklad - Třídění přímým vkládáním



Třídění přímým vkládáním – Insert Sort 1/2

Příklad - Třídění přímým vkládáním



Třídění přímým vkládáním – Insert Sort 1/2

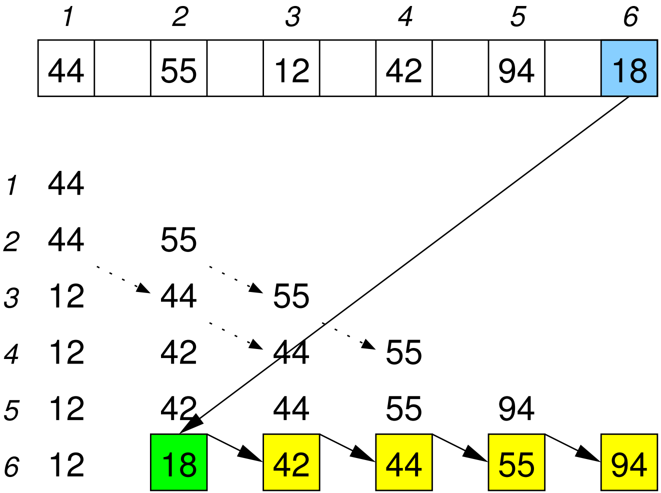
Příklad - Třídění přímým vkládáním

1	*2*	*3*	*4*	*5*	*6*
44	55	12	42	94	18

1	44				
2	44	55			
3	12	44	55		
4	12	42	44	55	
5	12	42	44	55	94
6					

Třídění přímým vkládáním – Insert Sort 1/2

Příklad - Třídění přímým vkládáním



Třídění přímým vkládáním – Insert Sort 2/2

for $i \in \langle 2, n \rangle$
„vlož a_i na patřičné místo mezi $a_1, \ldots, a_i$"

Příklad - Algoritmus

```
void insertSort(int[] array) {
   int x;
   int j;
   for (int i = 1; i < array.length; i++) {
      x = array[i];
      j = i - 1;
      while (j >= 0 && x < array[j]) {
         array[j + 1] = array[j];
          j--;
      }
      array[j + 1] = x;
   }
}
```

- Třídění binárním vkládáním - vkládání provádíme do již zatříděného pole $\langle a_1, \ldots, a_{i-1} \rangle$.

Třídění přímým vkládáním – Insert Sort 2/2

for $i \in \langle 2, n \rangle$

„vlož a_i na patřičné místo mezi $a_1, \dots, a_i$"

Příklad - Algoritmus

```
void insertSort(int[] array) {
   int x;
   int j;
   for (int i = 1; i < array.length; i++) {
      x = array[i];
      j = i - 1;
      while (j >= 0 && x < array[j]) {
          array[j + 1] = array[j];
           j--;
      }
      array[j + 1] = x;
   }
}
```

- Třídění binárním vkládáním - vkládání provádíme do již zatříděného pole $\langle a_1, \dots, a_{i-1} \rangle$.

Řazení přímým výběr – Select Sort 1/3

for $i \in \langle 1, n \rangle$ {
„najdi index k nejmenšího prvku v $\langle a_i, \ldots, a_n \rangle$,
$a_k = min\langle a_i, \ldots a_n \rangle$,
zaměň prvky a_i, a_k"
}

Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort

1		2		3		4		5		6
44		55		12		42		94		18

1

2

3

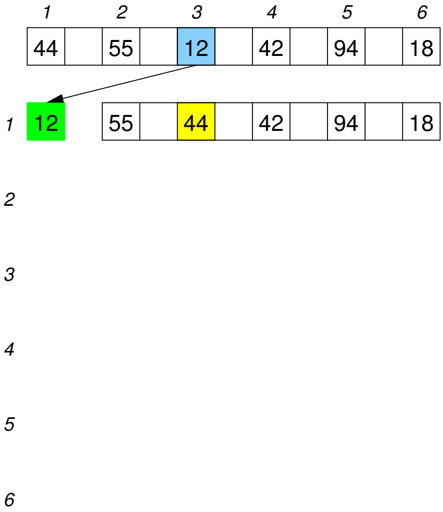
4

5

6

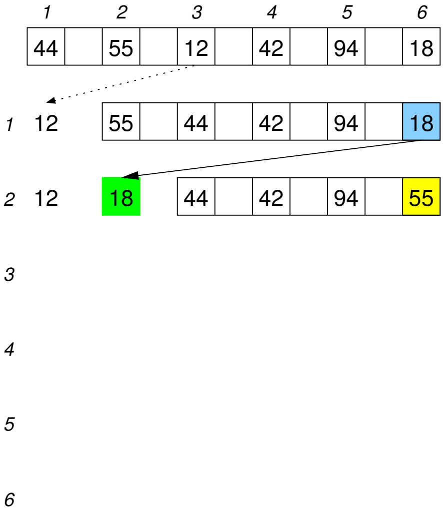
Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort



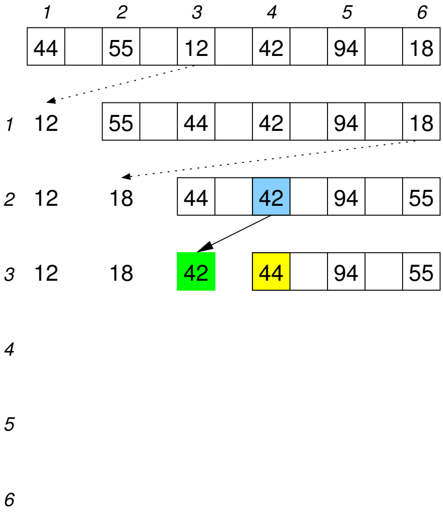
Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort



Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort



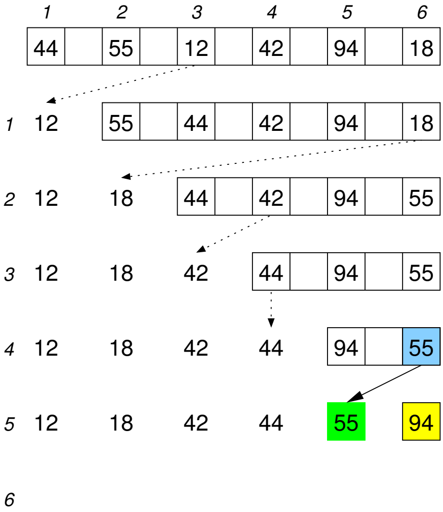
Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort

	1	2	3	4	5	6
	44	55	12	42	94	18
1	12	55	44	42	94	18
2	12	18	44	42	94	55
3	12	18	42	44	94	55
4	12	18	42	44	94	55
5						
6						

Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort



Řazení přímým výběr – Select Sort 2/3

Příklad - Řazení přímým výběrem Select Sort

	1	2	3	4	5	6
	44	55	12	42	94	18
1	12	55	44	42	94	18
2	12	18	44	42	94	55
3	12	18	42	44	94	55
4	12	18	42	44	94	55
5	12	18	42	44	55	94
6	12	18	42	44	55	94

Řazení přímým výběr – Select Sort 3/3

Příklad implementace

```
void selectSort(int[] array) {
   for (int i = 0; i < array.length-1; ++i) {
      int minIDX = i;
      for (int j = i + 1; j < array.length; ++j) {
         if (array[minIDX] > array[j]) {
            minIDX = j;
         }
      }
      swap(minIDX, i, array);
   }
}
```

Zde pro jednoduchost třídíme jen celá čísla, prakticky však zpravidla třídíme klíče datových položek.

- Složitost $O(n^2)$, kde n je počet položek

Řazení rozdělováním – Quick Sort

„Nejvýznamnější výměny jsou ty na velké vzdálenosti."

- Pole rozdělíme nějakým prvkem x (pivot).
- Nalevo od prvku x umístíme všechny prvky menší než x.
- Napravo od prvku x umístíme všechny prvky větší než x.
- Rozdělení opakujeme pro pole tvořené prvky nalevo od x a pro pole napravo od x.
- Postup opakujeme dokud nerozdělujeme jednoprvkové úseky.
- Strategie „rozděl a panuj."

Tuto strategii můžeme implementovat rekurzí

Rozdělení pole

Příklad - rozdělení

5	32	12	28	55	18	42	94

Rozdělení pole

Příklad - rozdělení

5	32	12	28	55	18	42	94

5	18	12	28	55	32	42	94

Řazení rozdělením – Quick Sort 1/2

```
void qsortPart(int l, int h, int[] array) {
   int i = l; //lower index
   int j = h; //higher index
   int pivot = array[l + (h - l) / 2];
   while (i <= j) {
      while(array[i] < pivot) {
         i++;
      }
      while(array[j] > pivot) {
         j--;
      }
      if (i <= j) {
         swap(i++, j--, array);
      }
   }
   if (l < j) {
      qsortPart(l, j, array);
   }
   if (i < h) {
      qsortPart(i, h, array);
   }
}
```

Řazení rozdělením – Quick Sort 2/2

```java
void quickSort(int[] array) {
   qsortPart(0, array.length-1, array);
}
```

- Složitost pro nejnepříznivější případ je $O(n^2)$
- Vhodně zvolený pivot výrazně snižuje časovou náročnost
- Randomizovaná varianta (vstupní pole permutováno a pivot je volen náhodně) – $O(n log_2 n)$

Řazení Select Sort vs Quick Sort

- Příklad spuštění Select Sort vs Quick Sort

```
javac DemoSort.java && java DemoSort 10
Values: 10 24 41 17 20 31 42 5 24 46
Values Select Sort: 5 10 17 20 24 24 31 41 42 46
Values Quick Sort:  5 10 17 20 24 24 31 41 42 46
```

```
java DemoSort 40000
Select Sort required time 3415 ms
 Quick Sort required time 10 ms
```

lec06/DemoSort.java

- Jednoduchý test výpočetní náročnosti můžeme udělat porovnáním výpočetních časů

  Např. využitím `System.currentTimeMillis`

- Skutečné výpočetní nároky se mohou lišit podle vstupních dat

  Zkuste zjistit počet operací pro vstupní pole setříděné sestupně

- Asymptotická složitost udává očekávané výpočetní nároky pro velké vstupy

  Reálné testování v tzv. „mikro benchmarks" může být zvláště v Javě zavádějící (HotSpot)

Zápis algoritmu Quick Sort

Implementace Quick Sort v programovacím jazyku Haskell

```
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++ qsort
     (filter (>= x) xs)
```

Pro zajímavost

Shrnutí přednášky

Diskutovaná témata

- Rekurze a rekurzivní algoritmy
- Příklady rekurzivních algoritmů:
  - Výpočet faktoriálu
  - Výpis posloupnosti čísel
  - Hanojské věže
  - Fibonacciho posloupnost
- Reprezentace množiny polem
  - Příklad implementace v algoritmu Eratosthenovo síto
- Rozklad na prvočinitele
- Příklad rekurze v řazení
- Příště: Objektově orientované programování v Javě