Algoritmy a datové struktury 1

2/2 Z+Zk, NTIN060

pomocné slajdy k algoritmickým technikám

rozděl a panuj, dynamické programování, pravděpodobnostní analýza

Pavel Veselý (IUUK)

vesely+ads1@iuuk.mff.cuni.cz

https://iuuk.mff.cuni.cz/~vesely/vyuka/ls24/ads1.html

Rozděl (problém) **a panuj** (nad algoritmy)

Divide et impera!

Kuchařka:

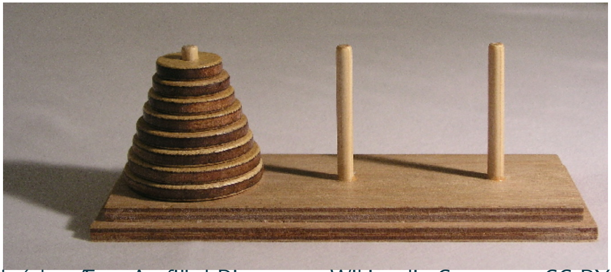
1. Rozdělíme problém na několik podproblémů

2. Pustíme algoritmus rekurzivně na každý podproblém

3. Spojíme řešení

Příklady:

- Hanojské věže



Autor obrázku: Ævar Arnfjörd Bjarmason, Wikimedia Commons, CC BY-SA 3.0

Rozděl (problém) **a panuj** (nad algoritmy)

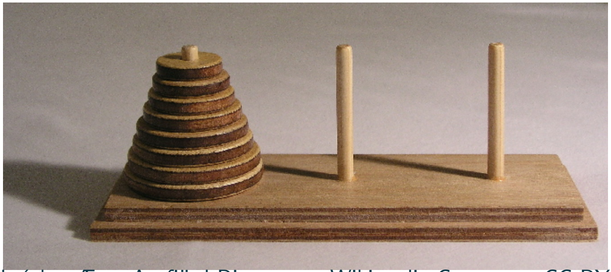
Divide et impera!

Kuchařka:

1. Rozdělíme problém na několik podproblémů
2. Pustíme algoritmus rekurzivně na každý podproblém
3. Spojíme řešení

Příklady:

- Hanojské věže



Autor obrázku: Ævar Arnfjörd Bjarmason, Wikimedia Commons, CC BY-SA 3.0

- MergeSort — třídění sléváním
- Karacubův algoritmus pro násobení čísel
- Strassenův algoritmus pro násobení matic
- QuickSelect pro hledání k-tého nejmenšího prvku (poslední přednáška)
- QuickSort pro třídění (poslední přednáška)

Násobení n-ciferných čísel

„Školní“ algoritmus v $\Theta(n^2)$:

```
      23958233
×         5830
―――――――――――――――
      00000000 ( =      23,958,233 ×     0)
     71874699  ( =      23,958,233 ×    30)
   191665864   ( =      23,958,233 ×   800)
+ 119791165    ( =      23,958,233 × 5,000)
―――――――――――――――
  139676498390 ( = 139,676,498,390        )
```

Násobení n-ciferných čísel

„Školní“ algoritmus v $\Theta(n^2)$:

```
      23958233
×         5830
―――――――――――――――
      00000000 ( =      23,958,233 ×     0)
     71874699  ( =      23,958,233 ×    30)
   191665864   ( =      23,958,233 ×   800)
+ 119791165    ( =      23,958,233 × 5,000)
―――――――――――――――
  139676498390 ( = 139,676,498,390        )
```

Lze násobit v čase $o(n^2)$?

Násobení n-ciferných čísel

„Školní“ algoritmus v $\Theta(n^2)$:

```
       23958233
×          5830
_______________
       00000000 ( =      23,958,233 ×     0)
      71874699  ( =      23,958,233 ×    30)
     191665864   ( =      23,958,233 ×   800)
+ 119791165     ( =      23,958,233 × 5,000)
_______________
  139676498390 ( = 139,676,498,390        )
```

Lze násobit v čase $o(n^2)$?

1960: Kolmogorov uspořádal workshop, kde chtěl ukázat dolní odhad $\Omega(n^2)$

- Karacuba na něm našel lepší algoritmus pomocí rozděl a panuj

Násobení n-ciferných čísel

„Školní“ algoritmus v $\Theta(n^2)$:

```
      23958233
×         5830
───────────────
      00000000 ( =      23,958,233 ×     0)
     71874699  ( =      23,958,233 ×    30)
   191665864   ( =      23,958,233 ×   800)
+ 119791165    ( =      23,958,233 × 5,000)
───────────────
  139676498390 ( = 139,676,498,390        )
```

Lze násobit v čase $o(n^2)$?

1960: Kolmogorov uspořádal workshop, kde chtěl ukázat dolní odhad $\Omega(n^2)$

- Karacuba na něm našel lepší algoritmus pomocí rozděl a panuj

Je násobení „asymptoticky těžší“ než sčítání?

Násobení n-ciferných čísel

„Školní“ algoritmus v $\Theta(n^2)$:

```
        23958233
×           5830
―――――――――――――――
        00000000 ( =      23,958,233 ×     0)
       71874699  ( =      23,958,233 ×    30)
     191665864   ( =      23,958,233 ×   800)
+ 119791165      ( =      23,958,233 × 5,000)
―――――――――――――――
    139676498390 ( = 139,676,498,390        )
```

Lze násobit v čase $o(n^2)$?

1960: Kolmogorov uspořádal workshop, kde chtěl ukázat dolní odhad $\Omega(n^2)$

- Karacuba na něm našel lepší algoritmus pomocí rozděl a panuj

Je násobení „asymptoticky těžší“ než sčítání?

- Pomocí *rozděl a panuj* lze dosáhnout pro každé $\varepsilon > 0$ složitosti $O_\varepsilon(n^{1+\varepsilon})$
- Pomocí Fourierovy transformace: $O(n \log n)$ (ADS 2)
- Schönhage a Strassen ('71): čas $O(n)$

Násobit lze v asymptoticky stejném čase jako sčítat!

Kuchařková věta (Master Theorem)

Věta: Rekurentní rovnice časové složitosti:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c) \qquad \text{a} \qquad T(k) = \Theta(1) \text{ pro } k = O(1)$$

má pro konstanty $a \geq 1, b > 1, c \geq 0$ řešení

- $T(n) = \Theta(n^c \log n)$, pokud $a/b^c = 1$
- $T(n) = \Theta(n^{\log_b a})$, pokud $a/b^c > 1$
- $T(n) = \Theta(n^c)$, pokud $a/b^c < 1$

Strassenův algoritmus pro násobení matic ('69)

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{21} \\ B_{12} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

Strassenův algoritmus pro násobení matic ('69)

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{21} \\ B_{12} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}); \\
M_2 &= (A_{21} + A_{22})B_{11}; \\
M_3 &= A_{11}(B_{12} - B_{22}); \\
M_4 &= A_{22}(B_{21} - B_{11}); \\
M_5 &= (A_{11} + A_{12})B_{22}; \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}); \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Zdroj obrázků: Wikimedia Commons, autor Cyp

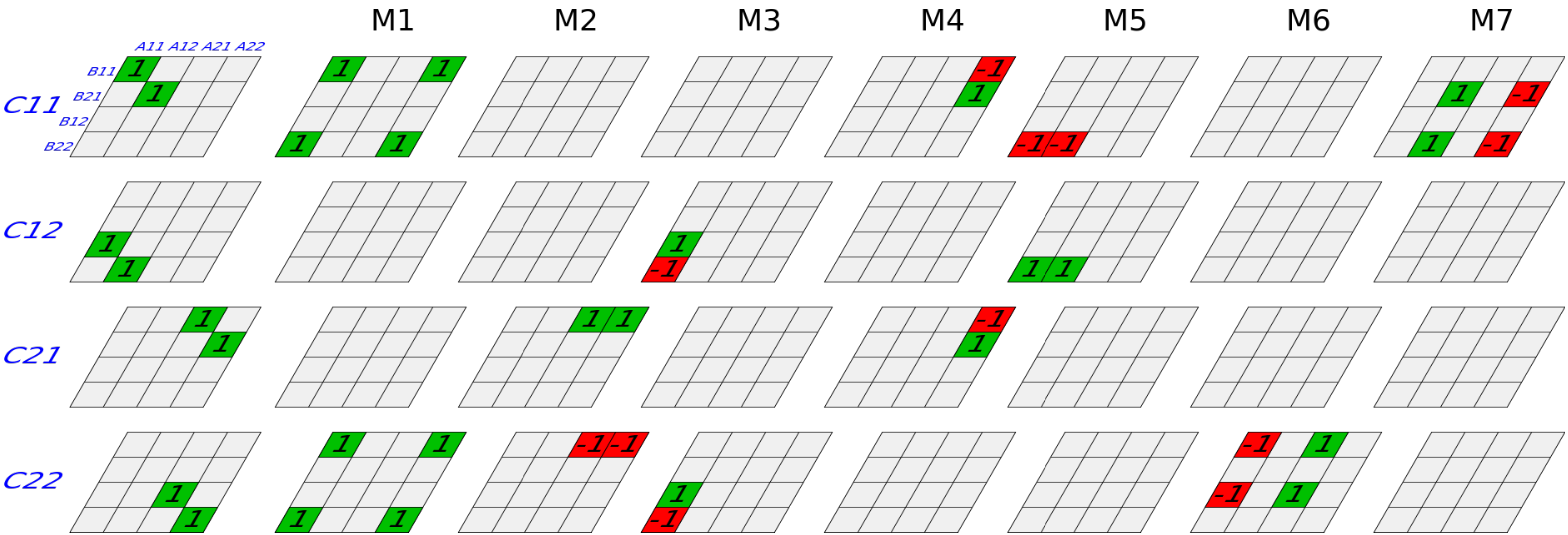
Strassenův algoritmus pro násobení matic ('69)

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{21} \\ B_{12} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}); \\
M_2 &= (A_{21} + A_{22})B_{11}; \\
M_3 &= A_{11}(B_{12} - B_{22}); \\
M_4 &= A_{22}(B_{21} - B_{11}); \\
M_5 &= (A_{11} + A_{12})B_{22}; \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}); \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Zdroj obrázků: Wikimedia Commons, autor Cyp



Strassenův algoritmus pro násobení matic ('69)

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{21} \\ B_{12} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}); \\
M_2 &= (A_{21} + A_{22})B_{11}; \\
M_3 &= A_{11}(B_{12} - B_{22}); \\
M_4 &= A_{22}(B_{21} - B_{11}); \\
M_5 &= (A_{11} + A_{12})B_{22}; \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}); \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Zdroj obrázků: Wikimedia Commons, autor Cyp

Konstanta ω: maticové násobení v čase $O(n^\omega)$ pro co nejmenší ω

- $\omega = \log_2 7 \approx 2.807$ [Strassen '69]

Strassenův algoritmus pro násobení matic ('69)

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{21} \\ B_{12} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}); \\
M_2 &= (A_{21} + A_{22})B_{11}; \\
M_3 &= A_{11}(B_{12} - B_{22}); \\
M_4 &= A_{22}(B_{21} - B_{11}); \\
M_5 &= (A_{11} + A_{12})B_{22}; \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}); \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Zdroj obrázků: Wikimedia Commons, autor Cyp

Konstanta ω: maticové násobení v čase $O(n^\omega)$ pro co nejmenší ω

- $\omega = \log_2 7 \approx 2.807$ [Strassen '69]
- ...
- $\omega < 2.37287$ [Le Gall '14]
- $\omega < 2.3728\mathbf{6}$ [Alman & Vassilevska Williams '21]
- $\omega < 2.371866$ [Duan, Wu & Zhou '23]
- $\omega < 2.371\mathbf{552}$ [Vassilevska Williams, Xu, Xu & Zhou '24]

Dynamické programování

- Vyvinul Bellman v 50. letech

Dynamické programování

- Vyvinul Bellman v 50. letech

Kuchařka:

1. Máme rekurivní algoritmus, ale s exponenciální časovou složitostí
   - např. pomocí Rozděl a panuj

2. Pozorování: probíhá mnoho opakovaných výpočtů

3. Řešení: pořídíme si tabulku na ukládání řešení podproblémů
   - kešování / memoizace

Dynamické programování

- Vyvinul Bellman v 50. letech

Kuchařka:

1. Máme rekurivní algoritmus, ale s exponenciální časovou složitostí
   - např. pomocí Rozděl a panuj
2. Pozorování: probíhá mnoho opakovaných výpočtů
3. Řešení: pořídíme si tabulku na ukládání řešení podproblémů
   - kešování / memoizace
4. Často lze odstranit rekurzi iterativním algoritmem

Dynamické programování

- Vyvinul Bellman v 50. letech

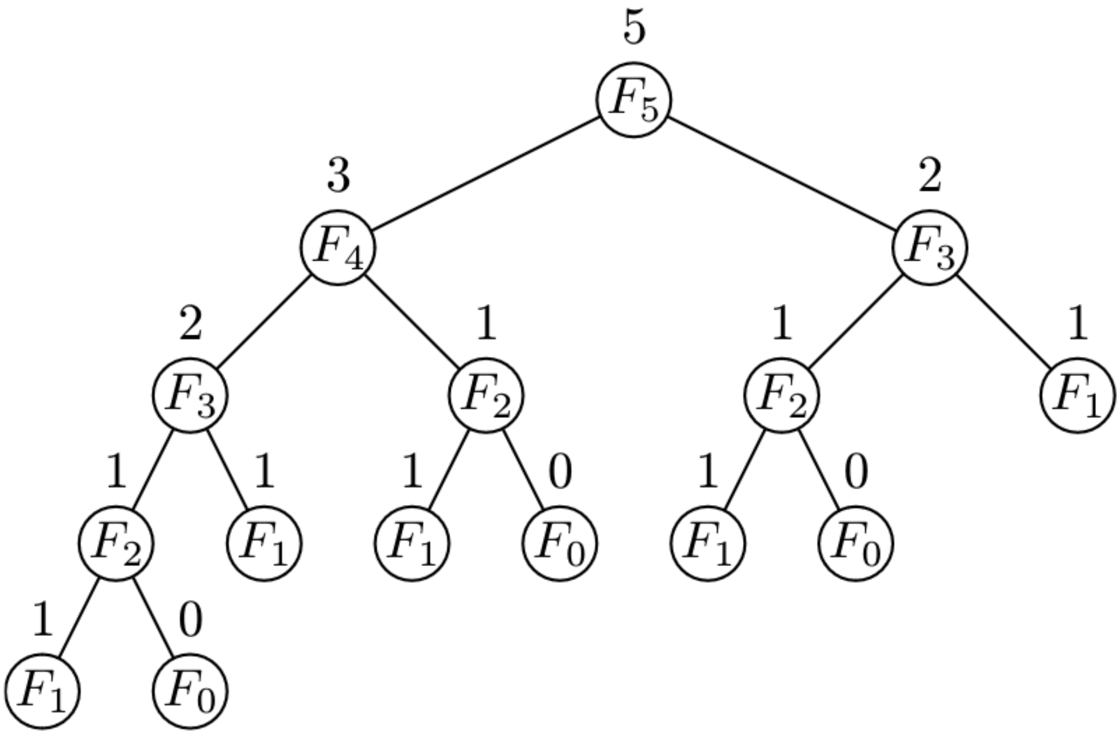
Kuchařka:

1. Máme rekurivní algoritmus, ale s exponenciální časovou složitostí
   - např. pomocí Rozděl a panuj
2. Pozorování: probíhá mnoho opakovaných výpočtů
3. Řešení: pořídíme si tabulku na ukládání řešení podproblémů
   - kešování / memoizace
4. Často lze odstranit rekurzi iterativním algoritmem

Příklady:

- Fibonacciho čísla
- Editační vzdálenost
- Nejdelší rostoucí podposloupnost
- Optimální vyhledávací stromy
- Nejdelší společná podposloupnost atd. (viz cvičení)

Fibonacciho čísla – strom rekurze



Obrázek 12.1: Rekurzivní výpočet Fibonacciho čísel

Editační (Levenštejnova) vzdálenost (opakování)

- Dány řetězce S a T nad konečnou abecedou.
- $\mathrm{ED}(S, T) =$ min. počet editačních operací, abychom z S dostali T
  - Editace: smazání, přidání nebo změna znaku

Editační (Levenštejnova) vzdálenost (opakování)

- Dány řetězce S a T nad konečnou abecedou.
- $\mathrm{ED}(S, T) =$ min. počet editačních operací, abychom z S dostali T
  - Editace: smazání, přidání nebo změna znaku

Pozorování:

- pokud $S_1 = T_1$, pak $\mathrm{ED}(S, T) = \mathrm{ED}(S[2:], T[2:])$
- pokud $S_1 \neq T_1$, pak $\mathrm{ED}(S, T) = \min \begin{cases} \mathrm{ED}(S[2:], T[2:]) + 1 & \text{(změna 1. znaku)} \\ \mathrm{ED}(S, T[2:]) + 1 & \text{(přidání 1. znaku)} \\ \mathrm{ED}(S[2:], T) + 1 & \text{(smazání 1. znaku)} \end{cases}$

Editační (Levenštejnova) vzdálenost (opakování)

- Dány řetězce S a T nad konečnou abecedou.
- $\mathrm{ED}(S, T) =$ min. počet editačních operací, abychom z S dostali T
  - Editace: smazání, přidání nebo změna znaku

Pozorování:

- pokud $S_1 = T_1$, pak $\mathrm{ED}(S, T) = \mathrm{ED}(S[2:], T[2:])$
- pokud $S_1 \neq T_1$, pak $\mathrm{ED}(S, T) = \min \begin{cases} \mathrm{ED}(S[2:], T[2:]) + 1 & \text{(změna 1. znaku)} \\ \mathrm{ED}(S, T[2:]) + 1 & \text{(přidání 1. znaku)} \\ \mathrm{ED}(S[2:], T) + 1 & \text{(smazání 1. znaku)} \end{cases}$

$\to$ rekurzivní algoritmus:

```
function EDREC(i, j)                    ▷ vrátí ED(S[i :], T[j :])
    if i > |S| then return |T| − j + 1
    if j > |T| then return |S| − i + 1
    if S_i = T_j then return EDrec(i + 1, j + 1)
    ℓ_z ← EDrec(i + 1, j + 1) + 1       ▷ změna znaku
    ℓ_p ← EDrec(i, j + 1) + 1           ▷ přidání znaku
    ℓ_s ← EDrec(i + 1, j) + 1           ▷ smazání znaku
    return min{ℓ_z, ℓ_p, ℓ_s}
```

Editační (Levenštejnova) vzdálenost (opakování)

- Dány řetězce S a T nad konečnou abecedou.
- $\mathrm{ED}(S, T) =$ min. počet editačních operací, abychom z S dostali T
  - Editace: smazání, přidání nebo změna znaku

Pozorování:

- pokud $S_1 = T_1$, pak $\mathrm{ED}(S, T) = \mathrm{ED}(S[2:], T[2:])$
- pokud $S_1 \neq T_1$, pak $\mathrm{ED}(S, T) = \min \begin{cases} \mathrm{ED}(S[2:], T[2:]) + 1 & \text{(změna 1. znaku)} \\ \mathrm{ED}(S, T[2:]) + 1 & \text{(přidání 1. znaku)} \\ \mathrm{ED}(S[2:], T) + 1 & \text{(smazání 1. znaku)} \end{cases}$

$\rightarrow$ rekurzivní algoritmus:

function $\textsc{EDrec}(i, j)$ $\triangleright$ vrátí $\mathrm{ED}(S[i:], T[j:])$
  if $i > |S|$ **then return** $|T| - j + 1$
  if $j > |T|$ **then return** $|S| - i + 1$
  if $S_i = T_j$ **then return** $\mathrm{EDrec}(i+1, j+1)$
  $\ell_z \leftarrow \mathrm{EDrec}(i+1, j+1) + 1$ $\triangleright$ změna znaku
  $\ell_p \leftarrow \mathrm{EDrec}(i, j+1) + 1$ $\triangleright$ přidání znaku
  $\ell_s \leftarrow \mathrm{EDrec}(i+1, j) + 1$ $\triangleright$ smazání znaku
  return $\min\{\ell_z, \ell_p, \ell_s\}$

Pozorování:

volámе pro $\leq (|S|+1) \cdot (|T|+1)$

různých parametrů

Nejdelší rostoucí podposloupnost (RPP)

- Dána posloupnost čísel $x_1, \ldots, x_n$

Nejdelší rostoucí podposloupnost (RPP)

- Dána posloupnost čísel $x_1, \ldots, x_n$
- $r(i) =$ maximální délka RPP začínající x_i

Nejdelší rostoucí podposloupnost (RPP)

- Dána posloupnost čísel $x_1, \ldots, x_n$
- $r(i)$ = maximální délka RPP začínající x_i

```
function RPPRec(i)
    d ← 1


    for all j = i + 1, ..., n do
        if x_j > x_i then
            d ← max(d, RPPrec(j) + 1)

    return d
```

Nejdelší rostoucí podposloupnost (RPP)

- Dána posloupnost čísel $x_1, \ldots, x_n$
- $r(i)$ = maximální délka RPP začínající x_i

```
function RPPRec(i)
    d ← 1
    if A[i] > 0 then
        return A[i]
    for all j = i + 1, ..., n do
        if x_j > x_i then
            d ← max(d, RPPrec(j) + 1)
    A[i] ← d
    return d
```

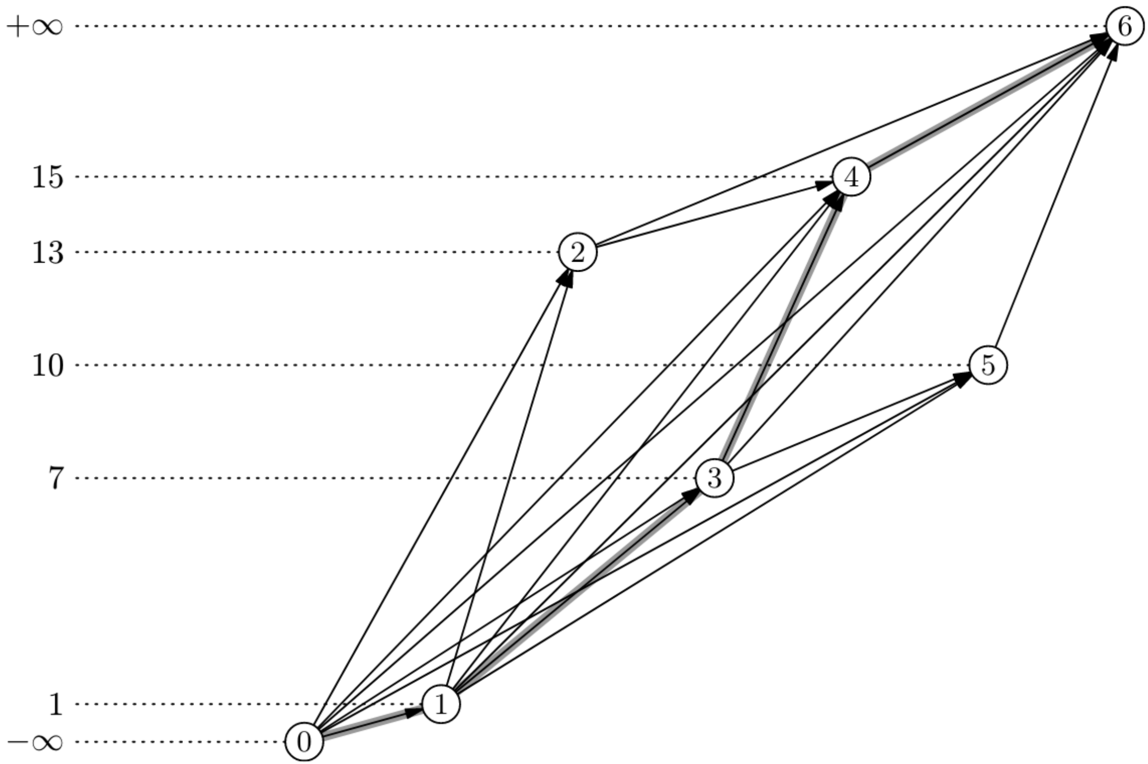
Nejdelší rostoucí podposloupnost (RPP)

- Dána posloupnost čísel $x_1, \ldots, x_n$
- $r(i) =$ maximální délka RPP začínající x_i

```
function RPPRec(i)
    d ← 1
    if A[i] > 0 then
        return A[i]
    for all j = i + 1, ..., n do
        if x_j > x_i then
            d ← max(d, RPPrec(j) + 1)
    A[i] ← d
    return d
```

```
function RPPIter(x_1, ..., x_n)
    x_0 ← −∞
    for all i = n, n − 1, ..., 0 do
        A[i] ← 1
        for all j = i + 1, ..., n do
            if x_i < x_j & A[i] < 1 + A[j] then
                A[i] ← 1 + A[j]
    return A[0] − 1        ▷ délka nejdelší RPP
```

Nejdelší rostoucí podposloupnost: grafový pohled



Obrázek 12.4: Graf reprezentující posloupnost
$-\infty, 1, 13, 7, 15, 10, +\infty$ a jedna z nejdelších cest

Optimální binární vyhledávací stromy (OptBVS)

Dána statická množina klíčů $k_1 < k_2 < \cdots k_n$ s váhami $w_1, \ldots, w_n$

Cíl: najít BVS T s minimální cenou

- Cena $T = \sum_{i=1}^{n} w_i \cdot h_i(T)$, kde $h_i(T) \geq 1$ je hloubka klíče k_i v T

Optimální binární vyhledávací stromy (OptBVS)

Dána statická množina klíčů $k_1 < k_2 < \cdots k_n$ s váhami $w_1, \ldots, w_n$

Cíl: najít BVS T s minimální cenou

- Cena $T = \sum_{i=1}^{n} w_i \cdot h_i(T)$, kde $h_i(T) \geq 1$ je hloubka klíče k_i v T
- Lze vyřešit v čase $\Theta(n^2)$ [Knuth '71]
  - Nechť $K[i,j] \in [i,j]$ je kořen OptBVS pro $k_i, \ldots, k_j$
  - Platí: $K[i,j-1] \leq K[i,j] \leq K[i+1,j]$
- $\Theta(n \log n)$ [Hu & Tucker '71]

Optimální binární vyhledávací stromy (OptBVS)

Dána statická množina klíčů $k_1 < k_2 < \cdots k_n$ s váhami $w_1, \ldots, w_n$

Cíl: najít BVS T s minimální cenou

- Cena $T = \sum_{i=1}^{n} w_i \cdot h_i(T)$, kde $h_i(T) \geq 1$ je hloubka klíče k_i v T
- Lze vyřešit v čase $\Theta(n^2)$ [Knuth '71]
  - Nechť $K[i,j] \in [i,j]$ je kořen OptBVS pro $k_i, \ldots, k_j$
  - Platí: $K[i,j-1] \leq K[i,j] \leq K[i+1,j]$
- $\Theta(n \log n)$ [Hu & Tucker '71]

Dynamické programování: obecný princip

máme systém podproblémů / stavů DP:

- je jich omezeně mnoho (polynomiálně)
- závislosti tvoří DAG $\Rightarrow$ lze je procházet v topologickém pořadí

Optimální binární vyhledávací stromy (OptBVS)

Dána statická množina klíčů $k_1 < k_2 < \cdots k_n$ s váhami $w_1, \dots, w_n$

Cíl: najít BVS T s minimální cenou

- Cena $T = \sum_{i=1}^{n} w_i \cdot h_i(T)$, kde $h_i(T) \geq 1$ je hloubka klíče k_i v T
- Lze vyřešit v čase $\Theta(n^2)$ [Knuth '71]
  - Nechť $K[i,j] \in [i,j]$ je kořen OptBVS pro $k_i, \dots, k_j$
  - Platí: $K[i,j-1] \leq K[i,j] \leq K[i+1,j]$
- $\Theta(n \log n)$ [Hu & Tucker '71]

Dynamické programování: obecný princip

máme systém podproblémů / stavů DP:

- je jich omezeně mnoho (polynomiálně)
- závislosti tvoří DAG $\Rightarrow$ lze je procházet v topologickém pořadí
- nutná podmínka: vlastnost optimální podstruktury
  - optimální řešení lze poskládat z optimálních řešení podproblémů

Hledání nejkratších cest mezi každou dvojicí vrcholů

Floydův-Warshallův algoritmus ('62; Roy '59)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)

Hledání nejkratších cest mezi každou dvojicí vrcholů

Floydův-Warshallův algoritmus ('62; Roy '59)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)

function $\textsc{Floyd-Warshall}(\text{matice } D^0)$
  $\triangleright$ $D^0_{u,v} = \ell(uv)$ pokud $uv \in E$, jinak $D^0_{u,v} = +\infty$
  for $k = 0, \ldots, n-1$ **do**
    for $u = 1, \ldots, n$ **do**
      for $v = 1, \ldots, n$ **do**
        $D^{k+1}_{u,v} \leftarrow \min\{D^k_{uv}, D^k_{u,k+1} + D^k_{k+1,v}\}$
  return D^n $\triangleright$ matice vzdáleností

Hledání k-tého nejmenšího a třídění za použití pravděpodobnosti

Dnes:

- QuickSelect
- QuickSort
- Pravděpodobnostní algoritmy vs. náhodné vstupy
- Dolní odhad na třídění
- Pokud zbyde čas: LinearSelect
  - k-tý nejmenší lineárně bez použití pravděpodobnosti

Hledání k-tého nejmenšího a třídění za použití pravděpodobnosti

Dnes:

- QuickSelect
- QuickSort
- Pravděpodobnostní algoritmy vs. náhodné vstupy
- Dolní odhad na třídění
- Pokud zbyde čas: LinearSelect
  - k-tý nejmenší lineárně bez použití pravděpodobnosti

function $\textsc{QuickSelect}((x_1, \ldots, x_n; k))$
    zvolme pivota p jako libovolné x_j
    $L \leftarrow$ prvky x_i t.ž. $x_i < p$
    $R \leftarrow$ prvky x_i t.ž. $x_i > p$
    $S \leftarrow$ prvky x_i t.ž. $x_i = p$
    if $|L| \geq k$ **then return** $\mathrm{QuickSelect}(L, k)$
    else if $|L| + |S| \geq k$ **then return** p
    else return $\mathrm{QuickSelect}(R, k - |L| - |S|)$

Hledání k-tého nejmenšího a třídění za použití pravděpodobnosti

Dnes:
- QuickSelect
- QuickSort
- Pravděpodobnostní algoritmy vs. náhodné vstupy
- Dolní odhad na třídění
- Pokud zbyde čas: LinearSelect
  - k-tý nejmenší lineárně bez použití pravděpodobnosti

function $\textsc{QuickSelect}((x_1, \ldots, x_n; k))$
    zvolme pivota p jako libovolné x_j
    $L \leftarrow$ prvky x_i t.ž. $x_i < p$
    $R \leftarrow$ prvky x_i t.ž. $x_i > p$
    $S \leftarrow$ prvky x_i t.ž. $x_i = p$
    if $|L| \geq k$ **then return** $\mathrm{QuickSelect}(L, k)$
    else if $|L| + |S| \geq k$ **then return** p
    else return $\mathrm{QuickSelect}(R, k - |L| - |S|)$

function $\textsc{QuickSort}((x_1, \ldots, x_n))$
    zvolme pivota p jako libovolné x_j
    $L \leftarrow$ prvky x_i t.ž. $x_i < p$
    $R \leftarrow$ prvky x_i t.ž. $x_i > p$
    $S \leftarrow$ prvky x_i t.ž. $x_i = p$
    $\ell_1, \ldots, \ell_{|L|} \leftarrow \mathrm{QuickSort}(L)$
    $r_1, \ldots, r_{|R|} \leftarrow \mathrm{QuickSort}(R)$
    return $\ell_1, \ldots, \ell_{|L|}, |S| \times p, r_1, \ldots, r_{|R|}$

Pravděpodobnostní algoritmy

Mají přístup k náhodným bitům

- Umíme vygenerovat rovnoměrně náhodné číslo z $\{0, \ldots, n\}$
- Časová složitost nebo výstup algoritmu jsou náhodné proměnné
- Příklad: QuickSelect a QuickSort s náhodnou volbou pivota
- Algoritmus je **deterministický**, pokud nepoužívá náhodné bity

Pravděpodobnostní algoritmy

Mají přístup k náhodným bitům

- Umíme vygenerovat rovnoměrně náhodné číslo z $\{0, \ldots, n\}$
- Časová složitost nebo výstup algoritmu jsou náhodné proměnné
- Příklad: QuickSelect a QuickSort s náhodnou volbou pivota
- Algoritmus je **deterministický**, pokud nepoužívá náhodné bity

Náhodné vstupy (stochastické)

- předpokládáme, že vstup je generován nějakou distribucí

Příklad: QuickSelect a QuickSort na náhodných vstupech

- vstup je náhodná permutace $\{1, \ldots, n\}$
- střední hodnota časové složitosti jako při náhodné volbě pivota
  - i při pevné volbě pivota, např. $x_{\lceil n/2 \rceil}$
- „průměrná složitost přes všechny vstupy“

Dolní odhady na třídění

Triviální: $\Omega(n)$

Lze třídit v $O(n)$:

- polynomiálně velká čísla: $x_i \leq n^c$ pro pevné c — RadixSort / číslicové třídění
- řetězce lexikograficky, pokud je abeceda konstantně velká

Dolní odhady na třídění

Triviální: $\Omega(n)$

Lze třídit v $O(n)$:

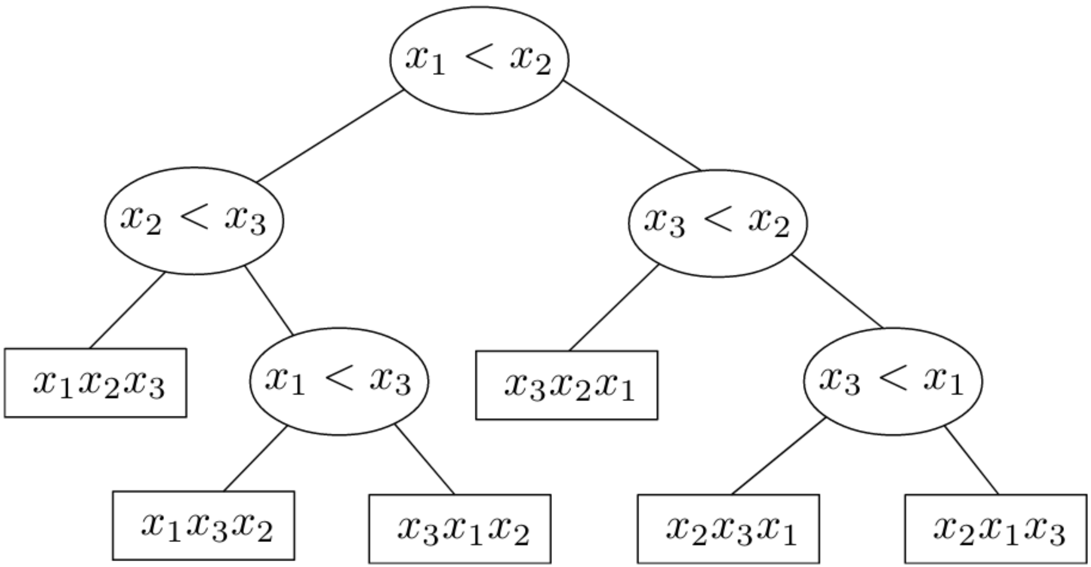
- polynomiálně velká čísla: $x_i \leq n^c$ pro pevné c — RadixSort / číslicové třídění
- řetězce lexikograficky, pokud je abeceda konstantně velká

Porovnávací model

- Prvky na vstupu lze pouze porovnat: pro jednoduchost jen $<$ a $=$
  - nejsou povoleny jiné operace s prvky, např. zápis čísla v soustavě s velkým základem
  - přesouvání/kopírování prvků v paměti povoleno
- Příklady:
  - InsertSort, BubbleSort: $\Theta(n^2)$
  - MergeSort, HeapSort: $\Theta(n \log n)$
  - QuickSort: $\Theta(n \log n)$ v průměru

Jde to lépe?

Příklad rozhodovacího strom pro třídění



Obrázek 3.1: Příklad rozhodovacího stromu pro 3 prvky