C2142 Návrh algoritmů pro přírodovědce

1. Od problému k algoritmu

Tomáš Raček

Jaro 2015

Problém, algoritmus

Problém – nerozřešená, sporná otázka, kterou je třeba řešit.

Příklady problémů:

- Kolik je třetí odmocnina z 27?
- Bude zítra pršet?
- $\hat{H}\Psi = E\Psi$
- Co si vzít na sebe do divadla?

Algoritmus je funkce mezi vstupem a výstupem reprezentována konečným způsobem v nějakém formalismu.

Ne každý problém lze algoritmicky řešit.

Potřeba formalismu

Přirozený jazyk trpí často nejednoznačností – dvojsmysly.

- Návrh putuje k Ústavnímu soudu v době, kdy je neúplný.
- Máme jen velvyslance prezidenta.

Formální jazyk má přesně danou syntaxi (strukturu) a sémantiku (význam).

- matematický zápis (př. $\forall x \in A : x \leq 0$)
- značkovací jazyky (př. HTML, XML,...)
- programovací jazyky (př. C/++, Python, Java, C#,...)

```
def fact(n):
    return n * fact(n - 1) if n > 0 else 1
```

Hledání počátku replikace jednoduchých bakterií

Biologický problém. Nalezněte část genomu, kde začíná replikace DNA (angl. origin of replication, oriC).

Způsoby řešení

- biolog – laboratoř
- informatik – počítač

Vstup: Genom (řetězec znaků A, C, G, T)

Výstup: Pozice počátku replikace v genomu.

Je toto informaticky správně formulovaný problém?

Problémy specifikace

Vstup: Genom (řetězec znaků A, C, G, T)

Výstup: Pozice počátku replikace v genomu.

Počátek replikace (oriC) není užitečně definován $\rightarrow$ potřeba dalších biologických informací.

Další vlastnosti:

- většinou několik stovek nukleotidů dlouhá oblast genomu
- obsahuje netriviální množství tzv. DnaA box (= krátké oblasti, na které se naváže DnaA protein a spustí tak replikaci)
  - typicky např. 9 nukleotidů
  - liší se pro každý organismus

Transformace problému

Problém nalezení oriC převedeme na hledání DnaA boxes. Oblast s jejich největším výskytem bude pak ukazovat na oriC.

(1) Zjednodušený problém. Nalezněte v textu řetězce délky **k** s nejvyšším počtem výskytů (DnaA box).

(2) Upravený problém. Nalezněte v textu řetězce délky **k** (DnaA box) v oblasti délky **n** ($\approx$ oriC) s minimálním počtem výskytů **t**.

Je náš zvolený přístup perspektivní? Pravděpodobnost, že existuje řetězec délky 9, který se vyskytuje v oblasti dlouhé 500 nukleotidů alespoň 3krát je asi 1/1300.

(1) Hledání nejčastějších slov v textu – řešení

Užitečným nástrojem pro řešení problému je dekompozice – rozložení problému na menší, lépe zvládnutelné jednotky.

PatternCount(text, pattern)

- pomocná funkce
- vrací počet výskytů řetězce **pattern** v řetězci **text**.

FrequentWords(text, k)

1. Pro každý podřetězec délky **k** řetězce **text** spočítej jeho výskyt pomocí funkce PatternCount
2. Urči nejvyšší nalezenou četnost
3. Vrať řetězce s touto nejvyšší četností

(1) Hledání nejčastějších slov v textu

```
def PatternCount(text, pattern):
    count = 0
    for i in range(0, 1  + len(text) - len(pattern)):
        if text[i : i + len(pattern)] == pattern:
            count += 1

    return count

def FrequentWords(text, k):
    counts = dict()
    frequentPatterns = set()

    for i in range(0, len(text) - k + 1):
        pattern = text[i : i + k]
        counts[pattern] = PatternCount(text, pattern)

    maxCount = max(counts.values())
    for (pattern, count) in counts.items():
        if count == maxCount:
            frequentPatterns.add(pattern)

    return frequentPatterns
```

Vybrané poznámky ze softwarového inženýrství

> ”Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”
>
> John Woods

Pro naše účely je forma uvedených algoritmů dostačující, v praxi je vhodné doplnit zejména:

- komentáře objasňující zvolený postup, případně myšlenkově obtížnější části algoritmu
- kontroly, zda jsou vstupní data v pořádku

(1) Upravená verze hledání nejčastějších slov

```
def FrequentWords(text, k):
    """ Find the most frequent words of the specified length"""
    counts = dict()
    frequentPatterns = set()

    if k < 1:
        raise Exception("The length of the pattern has\
                         to be at least 1")

    # Find the frequency for each pattern in the text
    for i in range(0, len(text) - k + 1):
        pattern = text[i : i + k]
        counts[pattern] = PatternCount(text, pattern)

    # Select the patterns with the highest occurrence
    maxCount = max(counts.values())
    for (pattern, count) in counts.items():
        if count == maxCount:
            frequentPatterns.add(pattern)

    return frequentPatterns
```

(2) Hledání nejčastějších slov v podřetězci

(2) Upravený problém. Nalezněte v textu řetězce délky **k** (DnaA box) v oblasti délky **n** (≈ oriC) s minimálním počtem výskytů **t**.

```
def FrequentWordsWithinRegion(text, k, n, t):
    frequentPatterns = set()

    for i in range(0, len(text) - n + 1):
        textRegion = text[i: i + n]
        counts = dict()

        for j in range(0, len(textRegion) - k + 1):
            pattern = textRegion[j : j + k]
            counts[pattern] = PatternCount(textRegion, pattern)

        for (pattern, count) in counts.items():
            if count >= t:
                frequentPatterns.add(pattern)

    return frequentPatterns
```

Korektnost

Korektnost algoritmu. Návrh/implementace odpovídá specifikaci (zadání).

Metody:

- testování *
- matematický důkaz
  - ověření, že korektní vstupní data budou algoritmem transformována na správná výstupní
- formální verifikace
  - ověření, zda model systému splňuje zadanou vlastnost
  - stejná míra jistoty jako u matematického důkazu
  - lze algoritmizovat
  - použitelná pouze pro velmi malé systémy

Testování

Testování. Levný a rychlý nástroj pro ověření základní funkcionality.

Co testovat?

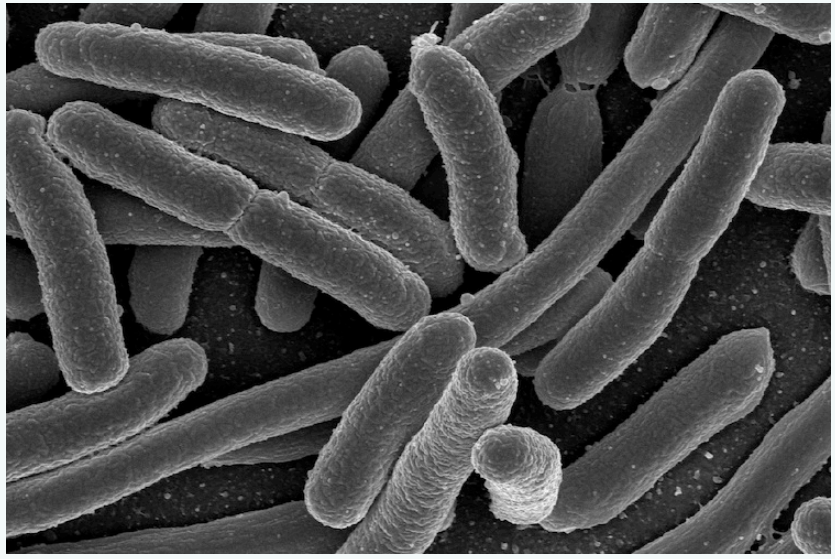
- obvyklý běh
- krajní hodnoty
- zakázané hodnoty

Motto:

> ”Testing shows the presence, not the absence of bugs.”
>
> Edsger W. Dijkstra

Proč tomu tak je?

Escherichia coli



Genom E. coli – asi 4,6 milionu nukleotidů (níže prvních 0,00313 %):

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAA
AAAAAGAGTGTCTGATAGCAGCTTCTGAACTGGTTACCTGCCGTGAGT
AAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATA
```

Praktický test

(1) Zahřívací problém. Nalezněte nejčastější řetězce délky 9 v rámci prvních 5000 nukleotidů genomu E. coli.

- použijeme funkci FrequentWords
- čas výpočtu asi 7 s

```
ACCATTACC TGGCGATGA CACCATTAC AACTGAAAG TGATGAAGA
```

(2) Kompletní genom E. coli Zopakujte (1) pro celý genom E. coli.

- téměř 1000krát větší problém než (1)
- odhad času výpočtu $1000 \cdot 7\ \text{s} = 7000\ \text{s} < 7200\ \text{s} = 2\ \text{h}$
- realita?

Vylepšení o další biologické poznatky 1

Náš vstupní řetězec představuje pouze jedno ze dvou navzájem reverzně komplementárních vláken genomu.

Komplementární báze:

- A $\leftrightarrow$ T
- C $\leftrightarrow$ G

Např. ReverseComplement(ACCCTG) = CAGGGT.

Závěr. Při hledání nejčastějších podřetězců je potřeba vzít do úvahy i jejich reverzní komplementy.

Vylepšení o další biologické poznatky 2

V řetězci DNA může dojít k mutacím (např. záměně jednoho nukleotidu za jiný).

Hammingova vzdálenost. Počet pozic, na kterých se dva řetězce stejné délky liší.

Př. Seznam všech řetězců (ze znaků A, C, G, T), které mají Hammingovu vzdálenost od řetězce AAA rovnu nejvýše 1:

```
GAA ACA CAA AAA AAC ATA AGA TAA AAG AAT
```

Závěr. Při hledání nejčastějších podřetězců zohledníme i jejich blízké (co do Hammingovy vzdálenosti) sousedy.