

PLIN015 Proseminář z počítačové lingvistiky II

Miloš Jakubíček



Centrum zpracování přirozeného jazyka
Fakulty informatiky, Masarykova univerzita
xjakub@fi.muni.cz

10. dubna 2014

Obsah

- 1 Organizační pokyny
- 2 Algoritmická složitost
- 3 Datové struktury
- 4 CQL
- 5 Syntax a syntaktická analýza

Cíl kursu

- navázat na PLIN013, obdobná skladba a cíle

Požadavky k zápočtu

- přiměřená účast (max. 3 neomluvené absence)
- vyhotovení 3 průběžných úkolů a složení závěrečného testu
- tentokrát zkusíme celkem 60 bodů ;)

Osnova

- ne nutně pevná, máte-li náměty, ozvěte se
- algoritmická složitost a její vazba na korpusové databáze pro ZPJ
- vyhledávání v korpusech, jazyk CQL
- syntaktická analýza, elementární algoritmy, ukázka gramatik

- cíl: změřit „výkon“, příp. „náročnost“ programu/algoritmu
- výsledky by měly být reprezentativní, odrážet skutečné vlastnosti algoritmu, ne jeho konkrétní chování na určitém typu HW
- klíčové otázky: **Co** měřit? **Jak** měřit?

Co měřit?

- čas – časová složitost: kolik času potřebuje program na své vykonání?
- paměť – prostorová/paměťová složitost: kolik paměti (úložného prostoru) program ke svému běhu potřebuje?
- obě vlastnosti jsou do určité míry komplementární – proč?
- budeme se zabývat především časovou složitostí

Jak měřit?

- Uvažujme následující posloupnosti, ve kterých chceme vyhledat zadaný prvek, např. 3
- 3 18 1
- 1 18 3
- 1 18 2 5 3
- 1 18 2 5 27 11 3
- 1 18 1 3 1 1
- 1 18 2 5 87 12 2 32 45 12 1 3
- Kdy nám to potrvá nejdéle? Na čem to závisí?

Asymptotická složitost

- cíl: vyjádřit složitost jako funkci vstupu (resp. jeho velikosti, délky), tedy říct, jak roste složitost algoritmu vzhledem k rostoucímu vstupu
- Můžeme uvažovat složitost v *nejlepším* případě, *průměrném* případě, *nejhorším* případě nebo *amortizovanou* složitost.
- 3 18 2 5 1
- 1 18 3 5 1
- 1 18 2 5 3
- V mnoha případech (ale ne vždy!) nás zajímá hlavně složitost v nejhorším případě.

Základní složitosti

seřazené od „nejmenší“, s užitím tzv. O-notace

název	složitost	$n = 10^2, c = 10$	$n = 10^6, c = 10$
konstantní	$O(1) = c$	10	10
logaritmická	$O(\log n)$	6,64	19,9
lineární	$O(n)$	100	1 000 000
lineárnělogaritmická	$O(n \cdot \log n)$	664	19 900 000
kvadratická	$O(n^2)$	10 000	10^{12}
kubická	$O(n^3)$	1 000 000	10^{18}
obecná polynomiální	$O(n^c)$	10^{20}	10^{600}
exponenciální	$O(c^n)$	10^{100}	$10^{1000000}$
faktoriálová	$O(n!)$	moc	strašně moc:)

Složitost problému vs. složitost algoritmu

Složitost algoritmu

Složitostí algoritmu rozumíme složitost konkrétní instance zadaného algoritmu (implementovatelného v nějakém programovacím jazyce). Algoritmy pracující s lepší než exponenciální/faktoriálovou složitostí označujeme jako *efektivní*.

Složitost problému

Složitostí problému rozumíme složitost *optimálního* algoritmu korektně řešícího zadaný problém.

Vyhledávání v neseřazené posloupnosti

- jaká je složitost demonstrovaného algoritmu? $O(n)$.
- jaká je složitost problému (tj. jde to lépe) a proč?

Vyhledávání v seřazené posloupnosti

- tzv. binárním vyhledáváním nebo-li půlením intervalů
- jaká je složitost demonstrovaného algoritmu? $O(\log_2 n)$.
- jaká je složitost problému (tj. jde to lépe) a proč?

Vztah asymptotické složitosti a výkonu HW

- uvažujme jeden krok výpočtu jako 1 s, 10 min odpovídá 600 krokům
- lineární algoritmus složitosti $3 \cdot n$: 10 min stačí na vykonání pro vstup velikosti 200
- exponenciální algoritmus 2^n : 10 min stačí na vykonání pro vstup velikosti 9 ($2^9 = 512$, $2^{10} = 1024$)
- uvažujme dvakrát vykonnější HW: jeden krok výpočtu = 0,5 s, 10 min odpovídá 1 200 krokům
- lineární algoritmus složitosti $3 \cdot n$: 10 min stačí na vykonání pro vstup velikosti 400
- exponenciální algoritmus 2^n : 10 min stačí na vykonání pro vstup velikosti 10 ($2^{11} = 2048$)

Algoritmy a datové struktury

- datová struktura: způsob uložení dat
- algoritmus: postup zpracování
- úzce propojené – datová struktura vynucuje algoritmus a obráceně
- \Rightarrow budeme mluvit o složitosti úkolu na konkrétní datové struktuře

Datové struktury

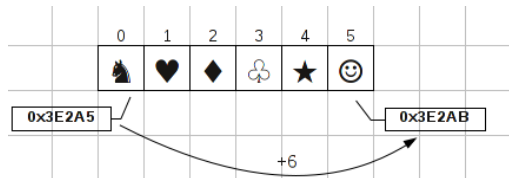
- ukázkové příklady: pole a lineární spojový seznam
- operace: přístup k n -tému prvku, vložení prvku, odstranění n -tého prvku

Indexování

- klíč – identifikátor záznamu (např. v relační databázi – „tabulce“), podle okolností jednoznačný či nikoli
- index – seřazená posloupnost hodnot pro jeden klíč, s ukazateli na záznam
- vybudování indexu \Rightarrow rychlé vyhledávání

Pole (array)

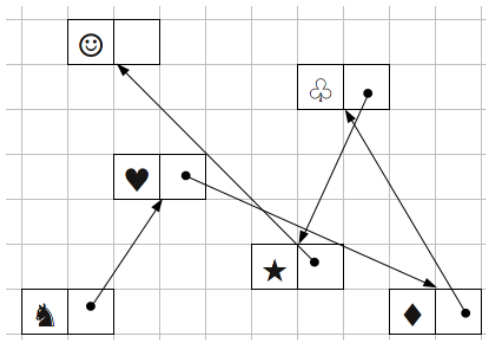
- uložení n hodnot v souvislé paměťové oblasti
- indexování číslem (\rightarrow adresou v paměti)
- prostorová složitost: $O(c \cdot n)$, kde n je počet záznamů, c velikost jednoho záznamu (konstanta)
- přístup k n -tému prvku: $O(1)$ (adresování v paměti)
- vložení prvku: $O(n)$ (realokace)
- odstranění prvku: $O(n)$ (realokace)



Seznam (list) I

- uložení n hodnot v nesouvislé paměťové oblasti
- s každou hodnotou je asociovaný ukazatel na následníka
- prostorová složitost: $O(c \cdot n)$, kde n je počet záznamů, c velikost jednoho záznamu (konstanta) – ale: záznam zde musí obsahovat i ukazatel na následníka \Rightarrow vyšší paměťové nároky než pole
- přístup k n -tému prvku: $O(n)$ (lineární průchod seznamem)
- vložení prvku: $O(1)$ (přepojení ukazatelů)
- odstranění prvku: $O(1)$ (přepojení ukazatelů)

Seznam (list) II



Shrnutí: pole vs. seznam

- pole: rychlejší přístup, pomalejší modifikace
- seznam: pomalejší přístup, rychlejší modifikace
- \Rightarrow vhodnost použití závisí na datech

Hashování

- v předchozích příklade bylo indexem vždy číslo
- můžeme ovšem chtít indexovat např. řetězcem („ke každému slovu přiřad' četnost“)
- nutná transformace nečíselné hodnoty na číslo – vytvoření tzv. *hashe* pomocí *hashovací funkce*

Hashovací funkce I

$h : D \rightarrow \mathcal{N}$, kde D je doména dat; požadavky:

- rychlá
- deterministická (pro stejná data vždy stejný výsledek)
- uniformní (výstupní hodnoty mají přibližně stejnou pravděpodobnost, všechny mají stejnou velikost)
- výstup volitelné délky
- obtížná reverzibilita (použití: hesla)
- minimální změna vstupu vyvolá velkou změnu výstupu

Hashovací funkci nazýváme perfektní, je-li její zúžení na konkrétních vstupních datech injektivní.

Hashovací funkce II

- obecně je hashovací funkce vždy neinjektivní a vznikají kolize
- důležitý požadavek: neměnnost vstupních dat po vytvoření hashe (*immutability*)
- Python: list = [], dict = {}

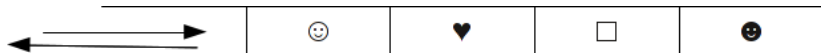
Fronta (queue, buffer)

- = datová struktura typu FIFO (First In First Out)
- implementace pomocí pole nebo spojového seznamu
- Python: `list.append()`, `list.pop(0)`, ale pomalé (`collections.deque`)!
- př. použití: prohledávání stromu do šířky



Zásobník (stack)

- = datová struktura typu LIFO (Last In First Out)
- implementace pomocí pole nebo spojového seznamu
- Python: `list.append()`, `list.pop()`
- př. použití: prohledávání stromu do hloubky



Binární halda (binary heap)

- datová struktura pro rychlé získávání minima (duálně maxima) bez třídění
- operace: vložení prvku, získání minima, odebrání minima (všechny $O(\log(n))$)
- http://en.wikipedia.org/wiki/Binary_heap

Řadící algoritmy

- mnoho různých algoritmů, zde pouze: bubblesort, quicksort, heapsort, mergesort
- složitost problému: $O(n \cdot \log(n))$
- http://en.wikipedia.org/wiki/Sorting_algorithm

Vyhledávání v korpusech – CQL

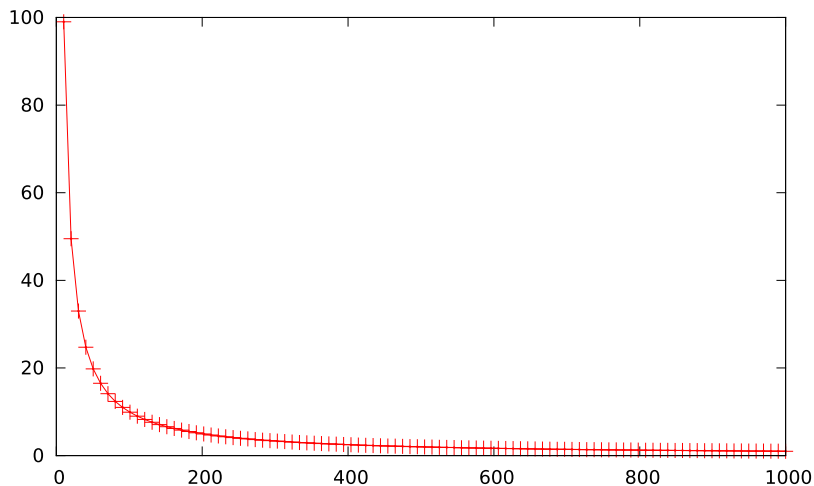
Korpus:

- poziční atributy – slovo, lemma, značka, ...
- struktury a strukturní atributy – dokument (autor, id, rok, ...), odstavec, věta
- vyhledávání: Manatee/Bonito/Sketch Engine
- <http://corpora.fi.muni.cz>
- <http://the.sketchengine.co.uk>

Indexace korpusu v Manatee

- tvorba tzv. lexikonu \Rightarrow očíslování řetězců \Rightarrow operace nad čísly místo nad řetězci
- operace pro převod: řetězec (`str`) – číslo (`id`) – pozice (`pos`)
- lexikon: `id2str`, `str2id`
- invertovaný (reverzní) index: `id2pos`
- text korpusu: `pos2id`
- tranzitivně lze i `pos2str`, `str2pos`
- základní myšlenka: seřazené proudy pozic a jejich slévání

Zipf's law I



Zipf's law II

- may be simplified to inductive definition:

Zipf's law (simplified)

frequency of the n -th element $f_n \approx \frac{1}{n} \cdot f_1$

- \Rightarrow frequency is inversely proportional to the rank according to frequency
- \Rightarrow one needs really large corpora to capture all the variety of many language phenomena

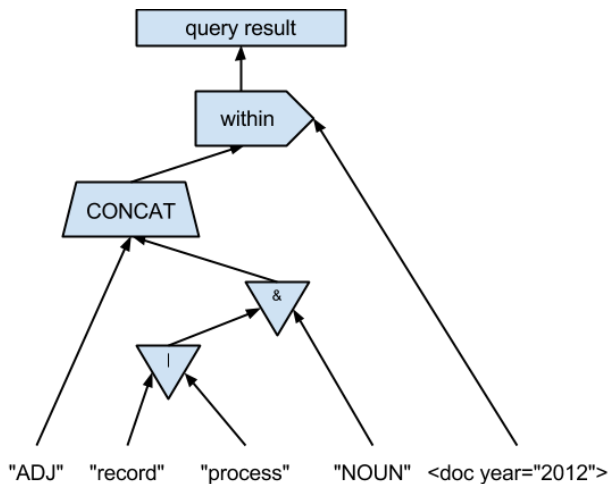
Zipf's law III

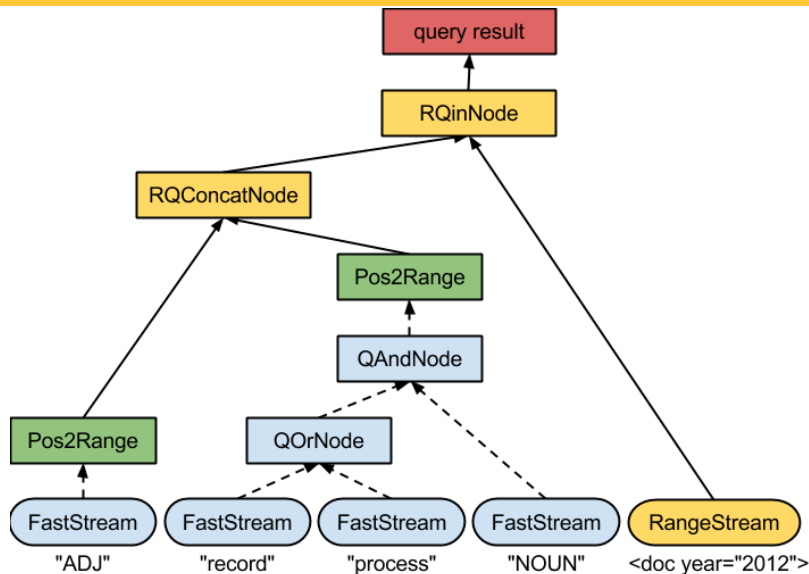
<u>tag</u>	<u>Freq</u>
NN	161881
NP	62669
NNS	56629
VVN	27545
VV	27481
VVD	27391
VVG	16922
VBD	13275
VBZ	11321
VVZ	8254
VVP	7912
VB	6377
VBP	5211
VHD	5190
VHZ	2497
VBN	2470
VHP	2445
VH	1780
NPS	1524
VBG	674
VHG	279
VHN	194

Substantives + Verb tags on the Brown corpus

Vyhodnocování CQL dotazu

Příklad: [tag="ADJ"] [(word="record" | word="process") & tag="NOUN"] within <doc year="2012"/>





Today's Corpora in SkE

- **LARGE** (= billions of tokens, and it's going to be worse)
- complex multi-level multi-value annotation
- wide range of languages
- growing demand on complex searching – moving from morphology to syntax and semantics
- search API for automatic information retrieval and post-processing in particular applications needed

CQL

- = Corpus Query Language (Christ and Schulze, 1994)
- positions and positional attributes: [attr="value"]
- structures and structural attributes: <str attr="value">>
- example:

```
[word=".*ing"& tag="V.*"]
  <doc id="20[5-9].*">
```

- established a within <str/> query:

```
[tag="N.*"]+ within <s/>
```

and alternative meet/union query:

```
(meet [lemma="take"] [tag="N.*"] -5 +5)
  (union (meet ...) (meet ...))
```

CQL in Manatee/Bonito

- enhancements and differences to the original CQL syntax
- `within <query>` and `containing <query>`
- `meet/union (sub)query`
- inequality comparisons
- frequency function

within/containing queries

- searching for particles:

```
[tag="PR.*"] within [tag="V.*"] [tag="ATO"]?  
[tag="AJ0"]* [tag="(PR.?|N.*)"] [tag="PR.*"]  
within <s/>
```

- searching for a Czech idiom “hnout někomu žlučí” (“to get somebody’s goat”):

word-by-word translated as:

hnout “move” [V, infinitive]

někomu “somebody” [N, dative]

žlučí “bile” [N, instrumental].

```
<s/> containing [lemma="hnout"] containing  
[tag=".*c3.*"] containing [word="žlučí"]
```

within/containing queries

- structure boundaries: begin: `<str>`, whole structure: `<str/>`, end: `</str>`
- **changes**: `within <str>` not allowed anymore, use `within <str/>`

meet/union queries

- combined with regular query: <s/>

```
containing (meet [lemma="have"] [tag="P.*"] -5 5)
```

```
containing (meet [tag="N.*"] [lemma="blue"])
```

- **changes**: meet/union queries can be used on any position, they can contain labels and no MU keyword is required (and deprecated):

```
(meet 1: [] 2: []) & 1.tag = 2.tag
```

Inequality comparisons

- former comparisons allowed only equality and its negation:
`[attr="value"]` `[attr!="value"]`
- inequality comparisons implemented: `[attr<="value"]`
`[attr>="value"]` `[attr!<="value"]` `[attr!>="value"]`
- intended usage:

`[tag="AJ.*"]` `[tag="NN.*"]` within `<doc year>="2009»`

- sophisticated comparison performed on the attribute value:
`<doc id<="CC20101031B»` matches e.g. BB20101031B,
CC20091031B, CC20101030B CC20101031A.

Fixed string comparisons

- normally the CQL values are regular expressions
- sometimes this is not desirable (batch processing needs escaping of metacharacters)
- new `==` and `!=` operator introduced for fixed strings comparison
- no escaping needed except for `"` and `'\'`
- examples: `"."`, `"$"`, `" "` matches a single dot, dollar sign and tilda, respectively, `"\n"` matches a backslash followed by the character `n`,

Frequency function

- a frequency constraint allowed in the global conditions part of CQL:

```
1: [tag="PP.*"] 2: [tag="NN.*"] & f(1.word) > 10
```

Performance evaluation

Tabulka: Query performance evaluation – corpora legend: ○ BNC (110M tokens), ● BiWeC (version with 9.5G tokens), * Czes (1.2G tokens)

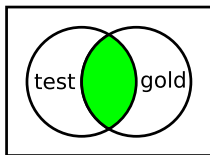
query	# of results	time (m:s)
○ [lemma="time"]	179,321	0.07
○ [lemma="t.*"]	14,660,881	3.12
○ Ex: particles	1,219,973	33.36
● Ex: particles	97,671,485	32:26.48
* Ex: idioms	66	1:6.86
○ Ex: meet/union	3	8.47
● Ex: meet/union	1457	7:13.12

Rozhraní systému Manatee

- podrobnější dokumentace k CQL: <http://trac.sketchengine.co.uk/wiki/SkE/CorpusQuerying>
- corpquery CORPUSNAME QUERY
- lsclex CORPUS ATTR
- lsslex CORPUS STRUCT ATTR

Precision, recall

- for classification problems in NLP, the standard evaluation is by means of precision and recall



$$\text{precision} = \frac{|\text{test} \cap \text{gold}|}{|\text{test}|} \quad \text{recall} = \frac{|\text{test} \cap \text{gold}|}{|\text{gold}|}$$

- two numbers, we just want to have one – F-score

$$F_1 \text{ score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F-score

- also F-measure
- general form: F_β score

$$F_\beta \text{ score} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 + \text{precision}) + \text{recall}}$$

- special case of $\beta = 1$ corresponds to the harmonic mean of precision and recall
- β can be used for favouring precision over recall (for $\beta < 1$) or vice versa (for $\beta > 1$)

Zkuste si

Na korpusu DESAM a czes vyzkoušejte:

- hledat výskyty přechodníků (přítomných i minulých)
- hledat jmenné fráze s genitivní vazbou
- hledat věty obsahující reflexivní slovesa
- hledat výskyty svého oblíbeného idiomu
- hledat shodné jmenné fráze do délky 4

Termín: do 21. 4. 2014, do odevzdávací vložte textový soubor s příslušnými dotazy v CQL

Exploiting CQL for word sketches

word sketch – one-page summary of word's collocational behaviour

- building a distributional thesaurus on top of word sketches
- using association metrics (scores) for measuring collocability of words

Word association scores I

■ T-Score

$$\frac{f_{AB} - \frac{f_A f_B}{N}}{\sqrt{f_{AB}}}$$

■ MI-Score

$$\log_2 \frac{f_{AB} N}{f_A f_B}$$

Church and Hanks, Word Association Norms, Mutual Information, and Lexicography, in Computational Linguistics, 16(1):22-29, 1990

■ MI³-Score

$$\log_2 \frac{f_{AB}^3 N}{f_A f_B}$$

Oakes, Statistics for Corpus Linguistics, 1998

Word association scores II

- minimum sensitivity

$$\min\left(\frac{f_{AB}}{f_B}, \frac{f_{AB}}{f_A}\right)$$

Pedersen, Dependent Bigram Identification, in Proc. Fifteenth National Conference on Artificial Intelligence, 1998

- MI.log-f (formerly called **salience**)

$$\text{MI-Score} \cdot \ln(f_{AB} + 1)$$

Kilgariff, Rychly, Smrz, Tugwell, "The Sketch Engine" Proc. Euralex 2004.

- Dice

$$\frac{2 \cdot f_{AB}}{f_A + f_B}$$

Measuring IAA

- naïve approach: count how many times people agreed on
- problem: it does not account for agreement by chance

Chance-corrected coefficients for IAA

- S (Benett, Alpert and Goldstein, 1954)
- π (Scott, 1955)
- κ (Cohen, 1960)
- (there is lot of terminology confusion, we follow Ron Artstein, Massimo Poesio: Inter-coder Agreement for Computational Linguistics, 2008)
- A_o – observed agreement
- A_e – expected (chance) agreement
- for all coefficients, they compute:

$$S, \pi, \kappa = \frac{A_o - A_e}{1 - A_e}$$

Chance-corrected coefficients for IAA

- S (Benett, Alpert and Goldstein, 1954)
 - assumes that all categories and all annotators have uniform probability distribution
- π (Scott, 1955)
 - assumes that different categories have different distributions shared across annotators
- κ (Cohen, 1960)
 - assumes that different categories and different annotators have different distributions
- devised for 2 annotators, various modifications for more than 2 annotators available

sed, diff, comm

- `sed` – zkratka z „stream editor“
- `sed 's/REGEX/NAHRADA/'` – provede náhradu regulárního výrazu `REGEX` za řetězec `NAHRADA`
- `diff` – zkratka z „difference“
- `diff SOUBOR1 SOUBOR2` – porovná soubory a vypíše nalezené rozdíly
- `comm` – zkratka z „compare“
- `comm -1 -2 -3 SOUBOR1 SOUBOR2` – vynechá řádky vyskytující se pouze v souboru 1 / 2 / obou (soubory musí být seřazené).
- `ajka`, `majka`, `desamb`

Gramatika

- \rightarrow PLIN004: $G = (\Sigma, N, P, S)$
- Chomského hierarchie, podle omezení na podobu pravidel ($A, B \in N, a \in \Sigma, \alpha, \beta \in (N \cup \Sigma)^*$):
 - typ 3: regulární gramatika: $A \rightarrow a, A \rightarrow aB,$
 - typ 2: bezkontextová gramatika: $A \rightarrow \beta,$
 - typ 1: kontextová gramatika: $\alpha \rightarrow \beta, |\alpha| \leq |\beta|,$
 - typ 0: frázová gramatika: $\alpha \rightarrow \beta.$
- pojetí gramatiky coby **generátoru** – regulární gramatika generuje regulární jazyk (a obdobně pro ostatní typy)
- potřeba „protikusu“ ke generátoru (syntetizátoru) – analyzátor
- \rightarrow GRAMATIKA – JAZYK – ANALYZÁTOR
- z PLIN004 již znáte: regulární gramatika – regulární jazyk (popsatelný regulárním výrazem) – konečný automat

Gramatika a analyzátor

- s postupnou relaxací omezení na tvar pravidel roste **vyjadřovací síla** gramatiky („složitost“ generovatelného jazyka)
- pochopitelně s tím rostou i nároky na analyzátor, zejména časová složitost analýzy (n značí délku vstupního řetězce):
 - regulární gramatika: $O(n)$ – proč?
 - bezkontextová gramatika: $O(n^3)$
 - kontextová gramatika: $O(n^6)$
- komplexní gramatické formalismy, některé zahrnující i sémantiku (LFG, HPSG, TAG, CCG, ...) – viz IB030

Přirozené jazyky a gramatiky

- Jak složité jsou přirozené jazyky?
- Lze přímočaře uplatnit Chomského hierarchii?
- Co je úkolem syntaktického analyzátoru přirozeného jazyka?
- Jaké jsou důsledky teorie formálních jazyků pro syntaktickou analýzu přirozeného jazyka?

Bezkontextová gramatika, zásobníkový automat

- bezkontextová gramatika = context-free grammar = CFG
- konečný automat = finite-state automaton = FSA
- zásobníkový automat = pushdown automaton = PDA
- podobně jako jazyk generovaný reg. gramatikou lze analyzovat FSA, můžeme jazyk generovaný CFG analyzovat pomocí PDA
- PDA: analýza shora dolů a zdola nahoru

ZPJ a teorie formálních jazyků

- skripta a slidy (s poděkováním) Shuly Wintner, University of Haifa: <http://cs.haifa.ac.il/~shuly/teaching/09/nlp/complexity-handout.pdf> a ve studijních materiálech
- skripta Černá-Kučera-Křetínský – FI:IB005 Formální jazyky a automaty: http://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf

Nedeterministická bezkontextová analýza

- shora dolů (Lookup-Rewrite)
 - na začátku analýzy obsahuje zásobník počáteční neterminál
 - operace: Lookup (najdi shodný prefix zásobníku a vstupu; odmaž – „přečti“), Rewrite (přepiš vrchol zásobníku podle pravidla gramatiky – nahraď pravou stranou)
 - nedeterminismus: vhodná volba pořadí operací Lookup-Rewrite
 - úspěšná analýza: prázdný zásobník, vstup byl celý přečten
- zdola nahoru (Shift-Reduce)
 - na začátku analýzy je zásobník prázdný
 - operace: Shift (vlož do zásobníku další slovo ze vstupu), Reduce (přepiš vrchol zásobníku podle pravidla gramatiky – nahraď levou stranou)
 - nedeterminismus: vhodná volba pořadí operací Shift-Reduce
 - úspěšná analýza: zásobník obsahuje právě počáteční neterminál, vstup byl celý přečten

Handling ambiguity

- Backtracking – try all options sequentially (usually degrades to factorial time complexity)
- Determinism – choose one option (preferably a good one) and keep to it
- Parallelism – try all options in parallel
- Underspecification – do not specify the ambiguous phenomenon

Deterministická analýza

- CKY / CYK (zdola nahoru, vyžaduje CNF)
- Earleyho parser (shora dolů)
- GLR (Generalized Left-to-right Right-most derivation parser) (zdola nahoru)
- Chart parser (zdola nahoru / shora dolů / řízený hlavou)
- → IB030

- složková analýza (phrase structure analysis)
- závislostní analýza (dependency analysis)
- co je cílem syntaktické analýzy?
- metody vyhodnocování (LAA, Parseval)

Příprava korpusu

- 1 Ujasnění obsahu korpusu (K čemu má sloužit? Jak získám vhodný text?)
- 2 Získání dat (Crawling, ne Google; wget, links)
- 3 Základní anotace (dokument, odstavce, specifické atributy)
- 4 Tokenizace (unitok.py, don't, A4)
- 5 Segmentace do vět (tecky.pl, MUDr., atd.)
- 6 Morfologická anotace / desambiguace (ajka / majka / desamb)
- 7 Další specifická anotace (např. syntaktická)
- 8 Zakódování (Manatee / encodevert; Corpus Architect)