PV030 Textové informační systémy

Petr Sojka

Fakulta informatiky
Masarykova univerzita v Brně

Jaro 2004

Část I

Informace o předmětu PV030

Úvodem

- Petr Sojka, **sojka@informatics.muni.cz**
- Konzultační hodiny jaro 2004: úterý 12:30–13:50 a čtvrtek 12:30–13:50, po domluvě emailem i jindy.
- Místnost B304, 3. patro bloku B, Botanická 68a.
- URL s materiály k předmětu:
  http://www.fi.muni.cz/~sojka/PV030/
- Rozdělení do cvičení (B204 → A107 vs. B311).

Určení a klasifikace předmětu

Prerekvizity předmětu:
U posluchače se předpokládají základní znalosti teorie formálních jazyků a automatů (IB005), zcela elementární základy teorie složitosti, znalosti programování a softwarových systémů.
Klasifikace:
Bodový systém sestává z ohodnocení práce v semestru (vnitrosemestrové písemky, které budou klasifikovány dohromady 30 body). Na závěr kurzu bude písemný test, na který je možno získat 70 bodů. Kromě toho je možno v průběhu semestru získat tzv. prémiové body. Klasifikační škála (změna vyhrazena) z/k/E/D/C/B/A odpovídá zisku 48/54/60/66/72/78/84 bodů.
Termíny závěrečného písemného testu budou 24. 5. v 15 hodin (D1) a 7. 6. 2004 ve 12 hodin (D1); opravný termín pak patrně 28. 6. 2004.

Předmět výuky

My books focus on timeless truth.
D. E. Knuth, Brno, 1996

Předmětem výuky tohoto předmětu je výklad základních principů, algoritmů a technik návrhu, vytváření a implementace textových informačních systémů (TIS)– ukládání (storage) a akvizice/vyhledávání (retrieval) informací.

Sylabus

① Základní pojmy informačních systémů. Klasifikace informačních systémů. Vyhledávací systémy.

② Vyhledávací algoritmy a datové struktury. Sousm. vyhl. metody s předzpracováním vzorků (MP, KMP, AC, RV).

③ Protism. vyhl. metody s předzpracováním vzorků (algoritmy BM, BMH, CW, BUC).

④ Vyhledávací metody s předzpracováním textu – indexové metody.

⑤ Metody indexování, konstrukce tezauru. Jak to dělal Google.

⑥ Vyhledávací metody s předzpracováním textu a vzorků – signaturové metody.

⑦ Jazyky pro vyhledávání.

⑧ Komprese dat, základní principy (entropie).

Sylabus (pokr.)

⑨ Statistické metody komprese dat.

⑩ Slovníkové metody komprese dat.

❶ Efektivní datové struktury pro uchování textů a slovníkových dat s využitím korpusové lingvistiky.

❷ Syntaktické metody. Kontextové modelování. Kontrola správnosti textu. Komprese textů s použitím neuronových sítí. Shlukování dokumentů a navigace v nich.

Knihy, skripta

- [MEL] Melichar, B.: *Textové informační systémy*, skripta ČVUT Praha, 2. vydání, 1996.
- [POK] Pokorný, J., Snášel, V., Húsek D.: *Dokumentografické informační systémy*, Karolinum Praha, 1998.
- [KOR] Korfhage, R. R.: *Information Storage and Retrieval*, Wiley Computer Publishing, 1997.
- [SMY] Smyth, B.: *Computing Patterns in Strings*, Addison Wesley, 2003.
- [KNU] Knuth, D. E.: *The Art of Computer Programming*, Vol. 3, Sorting and Searching, Second edition, 1998.
- [WMB] Witten I. H., Moffat A., Bell T. C.: *Managing Gigabytes: compressing and indexing documents and images*, Second edition, Morgan Kaufmann Publishers, 1998.

Doplňkové zdroje informací


[HEL] Held, G.: *Data and Image Compression*, Tools and Techniques, John Wiley & Sons, 4. vydání 1996.


[MEH] Melichar B., Holub J., A 6D Classification of Pattern Matching Problems, Proceedings of The Prague Stringology Club Workshop '97, Prague, July 7, CZ.


[GOO] Brin S., Page, L.: The anatomy of a Large-Scale Hypertextual Web Search Engine. WWW7/Computer Networks 30(1–7): 107–117 (1998).
`http://dbpubs.stanford.edu:8090/pub/1998-8`


[MeM] Mehryar Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing, *Natural Language Engineering*, 2(1):61–80, 1996.
`http://www.research.att.com/~mohri/cl1.ps.gz`

Doplňkové zdroje informací (pokr.)


[Sch] Schmidhuber J.: Sequential neural text compression, *IEEE Transactions on Neural Networks* 7(1), 142–146, 1996, **http://www.idsia.ch/~juergen/onlinepub.html**


[SBA] Salton G., Buckley Ch., Allan J.: Automatic structuring of text files, *Electronic Publishing* 5(1), p. 1–17 (March 1992). **http://columbus.cs.nott.ac.uk/compsci/epo/epodd/ep056gs.htm**


[WWW] stránky předmětu **~sojka/PV030/**, semináře DIS **http://www.inf.upol.cz/dis**, **http://nlp.fi.muni.cz/**, The Prague Stringologic Club Workshop 1996–2004 **http://cs.felk.cvut.cz/psc/**


Jones, S. K., Willett: *Readings in Information Retrieval*, Morgan Kaufman Publishers, 1997.

Doplňkové zdroje informací (pokr.)

Bell, T. C., Cleary, J. G., Witten, I. H.: *Text Compression*, Prentice Hall, Englewood Cliffs, N. J., 1991.

Storer, J.: *Data Compression: Methods and Theory*, Computer Science Press, Rockwille, 1988.

časopisy *ACM Transactions on Information Systems*, *Theoretical Computer Science*, *Neural Network World*, *ACM Transactions on Computer Systems*, *Knowledge Acquisition*.

`knihovna.muni.cz`, `umarecka.cz` (skripta Pokorný)

Část II

Základní pojmy TIS

TIS – motivace

$$\begin{array}{ccc} \text{realita} & \longleftrightarrow & \text{data} \\ \uparrow & & \uparrow \\ \text{potřeba informace} & \longleftrightarrow & \text{dotaz} \end{array}$$

- Abstrakce a mapování v informačním systému.
- Informační potřeba o realitě – dotazy nad daty.

Základní pojmy

Definice: ***Informační systém*** je systém, umožňující účelné uspořádání sběru, uchování, zpracování a poskytování informací.

Definice: ***Ektosystém*** se skládá z uživatelů IS, investora IS a toho, kdo systém provozuje (user, funder, server). Na příkladě IS MU jsou to uživatelé IS, MU reprezentovaná kvestorem a CVT a ÚVT MU. Ektosystém není pod kontrolou designéra IS.

Definice: ***Endosystém*** sestává z použitého hardware (media, devices), a software (algorithms, data structures) a je plně pod kontrolou designéra IS.

Požadavky na TIS

Různost hledisek na TIS

- efektivita (user)
- ekonomika (funder)
- efektivnost (server)

a z toho vyplývající kompromisy. Naše hledisko bude hledisko architekta TIS respektujícího požadavky ektosystému IS.
Pro problematiku ektosystému IS viz PV045 Management IS.

Od dat k moudrosti

- Údaj – konkrétní vyjádření zprávy ve formě posloupnosti symbolů nějaké abecedy.
- Informace – odraz poznaného nebo předpokládaného obsahu skutečností. Informace závisí na subjektu, jemuž je určena. Hlediska:
  - kvantitativní (teorie informace);
  - kvalitativní (význam, sémantika);
  - pragmatické (hodnotové – význam, užitečnost, upotřebitelnost, periodicita, aktuálnost, hodnověrnost);
  - ostatní (pohotovost, podrobnost, úplnost, jednoznačnost, dostupnost, náklady na získání).
- Znalost (knowledge).
- Moudrost (wisdom).

Informační proces

Definice: ***Informační proces*** – proces vzniku informací, jejich zobrazení ve formě údajů, zpracování, poskytování a využití. Tomuto procesu odpovídají operace nad informacemi.

Data/signály ⟶ Informace ⟶ Znalost ⟶ Moudrost.

Klasifikace IS podle převládající funkce

① dokumentografické vyhledávací systémy (Information Retrieval Systems).

② databázové systémy (DMBS), relační DB (PB154, PB155, PV003, PV055, PV136, PB114).

③ faktografické systémy pro řízení (Management Information Systems PV045).

④ systémy pro podporu rozhodování (Decision Support Systems PV098).

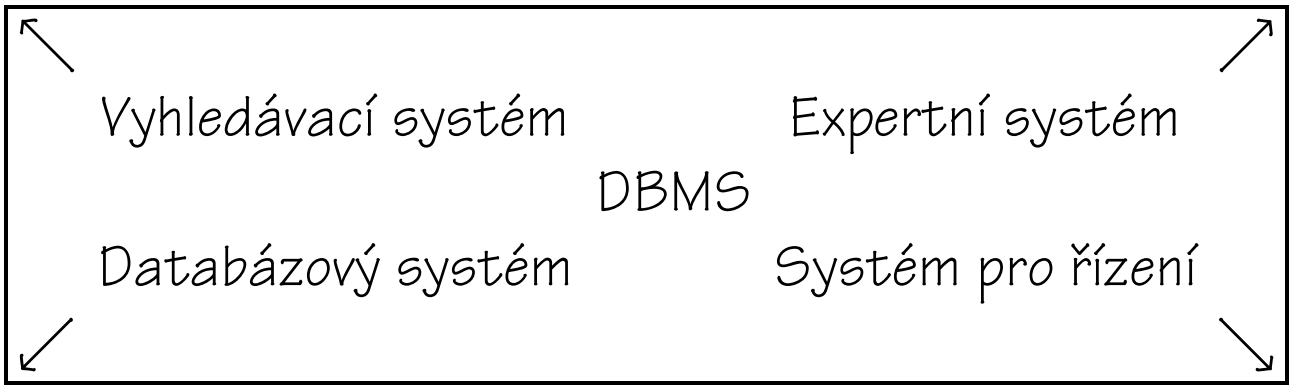
⑤ expertní (Expert Systems), dotazovací (Question Answering Systems) a znalostní (Knowledge-Based) systémy, (PA031).

Klasifikace IS podle převládající funkce (pokr.)

⑥ specifické informační systémy (geografické PV019, PA049, PA050, medicínské PV048, enviromentální PV044, podnikové PV043, ve státní správě PV058, PV059, knihovnické PV070); dále též PV063 Aplikace databázových systémů.

Související oblasti vyučované na FI:
Technologie informačních systémů: PA102, PA105.
Specifické aspekty indexování: Indexování multimediálních dat: PA128.
Implementace databázových systémů: PA152.

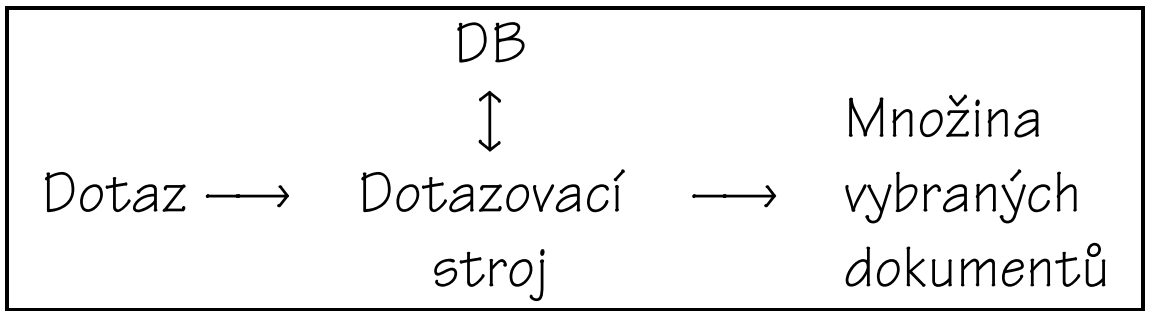
Různost pohledů na TIS



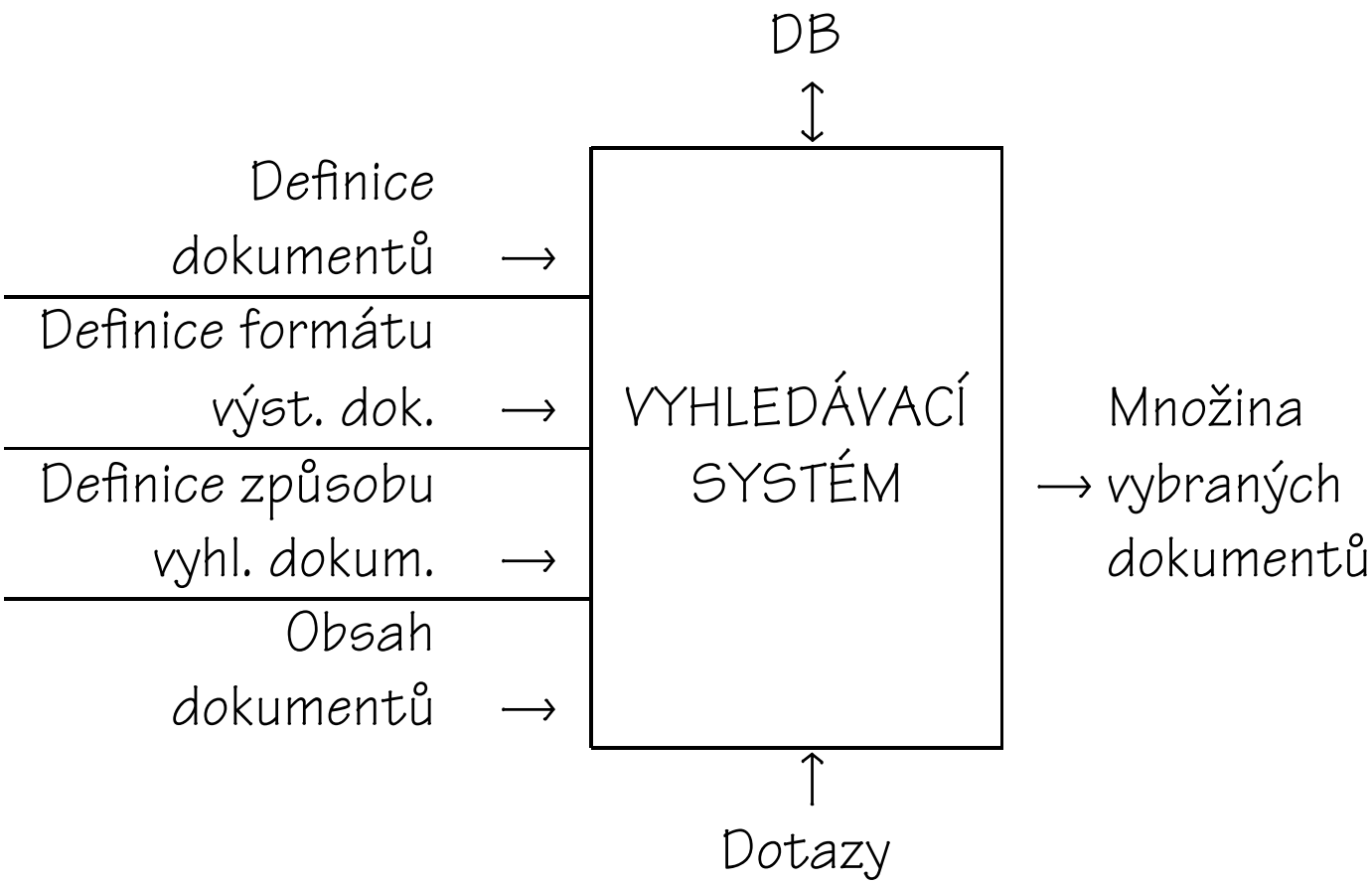
Minianketa

1. Co očekáváte od tohoto předmětu? Co byste se zde chtěli dozvědět? Vyhovuje sylabus?
2. Co neočekáváte (čemu byste se raději vyhnuli)?
3. Jaké příbuzné předměty jste absolvoval/hodláte absolvovat?
4. Použití IS (z hlediska uživatele)
   a) Jaké (T)IS používáte?
   b) Rozsah? Kolik hledání v (T)IS provádíte za měsíc?
   c) Jak jste spokojeni?
5. Vytváření IS (server)
   a) Jaký (T)IS nebo jeho komponentu jste realizoval? Oblast, rozsah?
   b) Jak jste s tímto (T)IS spokojeni, slabá místa?

Vyhledávací systémy (VS) – princip



Prázdný vyhledávací systém



Vyhledávací systémy – klasifikace

① Klasifikace dle směru průchodu textem: sousměrné/protisměrné

② Klasifikace dle (před)zpracování textu a vzoru:

- ad fontes (hledání v samotném textu)
- text surrogate (hledání v náhražce textu)
- náhražky:
  index: uspořádaný seznam významných prvků s odkazy do původního textu;
  signatura: řetězec příznaků indikující přítomnost významných prvků v textu

Vyhledávací systémy – klasifikace (pokr.)

		předzpracování textu	
		ne	ano
předzpracování vzorků	ne	I	III
	ano	II	IV

I – elementární algoritmy

II – vytvoření vyhledávacího stroje

III – indexové metody

IV – signaturové metody

Vyhledávání – formulace problému

Klasifikace dle kardinality množiny vzorů:

① Vyhledej jeden vzorek V v textu T. Výsledek: ano/ne.

② Vyhledej konečnou množinu vzorků $P = \{v_1, v_2, \ldots, v_k\}$. Výsledek: informace o tom, kde se v textu vyskytuje některý ze zadaných vzorků.

③ Vyhledej nekonečnou množinu vzorků zadanou regulárním výrazem R. R definuje potenciálně nekonečnou množinu $L(R)$. Výsledek: Informace, kde se v textu vyskytuje některý ze vzorků z $L(R)$.

Alternativy formulace problému vyhledávání:

a) první výskyt.

b) všechny výskyty bez překrývání.

c) všechny výskyty včetně překrývajících se.

Vyhledávání – formalizace problému

Řetízek korálků. Korálky ⟶ element. Číslování elementů přirozenými čísly. Ne nutně čísla, ale návěští.

0) Každý element má návěští, které je jedinečné.
1) Každý element s návěštím x (s výjimkou nejlevějšího) má jednoznačného předchůdce označovaného $pred(x)$.
2) Každý element s návěštím x (s výjimkou nejpravějšího) má jednoznačného následníka označovaného $succ(x)$.
3) Pokud element x není nejlevější, $x = succ(pred(x))$.
4) Pokud element x není nejpravější, $x = pred(succ(x))$.
5) Pro každé různé elementy x a y existuje kladné k tak, že buď $x = succ^k(y)$ nebo $x = pred^k(y)$.

Vyhledávání – formalizace problému (pokr.)

Pojem zřetězení:

Definice: **Řetězec** je množina elementů splňujících pravidla 0)–5).

Definice: **Lineární řetězec**: Řetězec, který má konečně mnoho prvků včetně nejlevějšího a nejpravějšího.

Definice: **Náhrdelník**.

Definice: **Abeceda A**. **Písmena abecedy**. **A^+**. **Prázdný řetěz ε**. **$A^* = A^+ \cup \{\varepsilon\}$**.

Definice: **Lineární řetězec nad A**: prvek A^+.

Definice: **Vzorek**. **Text**.

Část III

Přesné vyhledávání

Naivní vyhledávání, vyhledávání hrubou silou, elementární vyhledávací algoritmus

```
proc Brute-Force-Matcher(VZOREK,TEXT):
T:=length[TEXT]; V:=length[VZOREK];
for i:=0 to T-V do
  if VZOREK[1..V]=TEXT[i+1..i+V]
  then print "Vzorek nalezen na pozici i";
```

Analýza časové složitosti naivního vyhledávání

- složitost měřena počtem porovnání, délka vzorku V, délka textu T.
- horní odhad $S = V \cdot (T - V + 1)$, tedy $O(V \times T)$.
- nejhorší příklad VZOREK $= a^{V-1}b$, TEXT $= a^{T-1}b$
- přirozené jazyky: (průměrná) složitost (počet porovnání) výrazně menší, neboť nedochází ke shodě počátků slov příliš často. Pro angličtinu $S = C_E \cdot (T - V + 1)$, C_E empiricky naměřeno 1.07, t. j. prakticky lineární.
- C_{CZ}? C_{CZ} vs. C_E?
- urychlení? aplikace pro více vzorků? nekonečný počet?
- verze algoritmu (S, Q, Q') na cvičení.

Vyhledávací sousměrné metody

Při předzpracování je prozkoumána struktura vzorku a na jejím základě vytvořen vyhledávací algoritmus (stroj).

Definice: **Přesné** (vs. **proximitní**) vyhledávání cílí k přesnému nalezení vzorku.

Definice: **Sousměrné** (vs. **protisměrné**) vyhledávání porovnává vzorek zleva doprava (vs. zprava doleva).

Sousměrné metody

1. 1 vzorek:
   - Shift-Or algoritmus.
   - Knuth-Morris-Prattův algoritmus, (KMP, nalezen (MP) 1970, publikován 1977).
   - Karp-Rabinův algoritmus, (KR, 1987).
2. n vzorků: Aho-Corasickové algoritmus, (AC, 1975).
3. ∞ vzorků: konstrukce vyhledávacího stroje (KA) pro vyhledávání nekonečné množiny vzorků (regulární výrazy).

Shift-Or algoritmus

- Vzorek $v_1 v_2 \ldots v_m$ nad abecedou $\Sigma = a_1, \ldots, a_c$.
- Incidenční matice X $(m \times c)$, $X_{ij} = \begin{cases} 0 & \text{pokud } v_i = a_j \\ 1 & \text{jinak.} \end{cases}$
- Sloupec matice X odpovídající znaku a_j označme A_j.
- Na začátku do R jednotkový sloupec. V každém kroku algoritmu se řetězec R posune o jednu pozici dolů (horní pozice se naplní nulou) a přečte se ze vstupu jeden znak a_j. Výsledek posunu se zkombinuje s řetězcem A_j pomocí binární disjunkce: $R := \mathrm{SHIFT}(R)\ \mathrm{OR}\ A_j$.
- Algoritmus skončí úspěšně pokud se na dolní pozici R dostane 0.

Shift-Or algoritmus (pokr.)

Příklad: $V = vzorek$ nad $\Sigma = \{e, k, o, r, v, z\}$.
Srv. [POK, strana 31–32].

Naivní vyhledávání – algoritmy

Vyjádřete časovou složitost následujících vyhledávacích algoritmů pomocí proměnných c a s, kde c je počet testů a platí:

- je-li nalezen index i, pak $c = i$ a $s = 1$
- není-li nalezen, pak $c = T$ a $s = 0$

Naivní vyhledávání – algoritmus S

```
     vstup: var TEXT : array[1..T] of slovo;
     VZOREK : slovo;
     výstup (v proměnné FOUND): Ano/Ne
1    I:=1;
c    while I≤ T do
     begin
c    if TEXT[I]=VZOREK then break;
c-s  inc(I);
     end;
2    FOUND:=(I≤T);
```

Nalevo od příkazů je uvedena jejich složitost.

Celková časová složitost tedy je $O(T) = 3c - s + 3$.

Maximální složitost (která se běžně uvádí) je $O(T) = 3T + 3$.

Algoritmus Q aneb jak je tomu s použitím zarážek

```
    vstup: var TEXT : array[1..T+1] of slovo; VZOREK : slovo;
    výstup (v proměnné FOUND): Ano/Ne
1   I:=1;
1   TEXT[T+1]:=VZOREK;
c   while TEXT[I]<>VZOREK do
c-1     inc(I);
2   FOUND:=(I<>T+1)
```

Index je v tomto případě vždy nalezen; proto je v předposledním řádku algoritmu uvedena složitost $c-1$ místo $c-s$ (ačkoliv jsou shodné). Dále je potřeba si uvědomit, že maximální možná hodnota c je o 1 větší než v předchozím algoritmu (ale uvádět $c+1$ místo c by nebylo správné). Celková složitost $O(T) = 2c + 3$. Maximální složitost: $O(T) = 2T + 5$.

Algoritmus Q' aneb jak je tomu s použitím expanze cyklu

vstup: `var TEXT : array[1..T+1] of slovo;`
`VZOREK : slovo;`
výstup (v proměnné FOUND): Ano/Ne

1	`I:=1;`
1	`TEXT[T+1]:=VZOREK;`
$\lceil c/2 \rceil$	`while TEXT[I]<>VZOREK do`
	`begin`
$\lfloor c/2 \rfloor$	`if TEXT[I+1]=VZOREK then break;`
$\lfloor (c-1)/2 \rfloor$	`I:=I+2;`
	`end;`
3	`FOUND:=(I<T)or(TEXT[T]=VZOREK);`

Celková složitost: $O(T) = c + \lfloor (c-1)/2 \rfloor + 5$. Maximální složitost: $O(T) = T + \lfloor T/2 \rfloor + 6$. Podmínka v závěru algoritmu zajišťuje jeho funkčnost (není to ale jediná možnost, jak vyřešit inkrementaci cyklu o 2).

Část IV

Přesné sousměrné vyhledávání jednoho vzorku

Osnova (Týden druhý)

1. Zhodnocení ankety, administrativní poznámky.
2. Přesné vyhledávací metody II (s předzpracováním vzorku, sousměrné): KMP (animace), Rabin-Karp, AC.
3. Vyhledávací stroj.
4. Vyhledávání pomocí konečných automatů.
5. Vyhledávání nekonečné množiny vzorků.

Zhodnocení úvodní ankety

① Ano: sylabus vyhovuje; kladně hodnoceno pitvání Google; indexace a vyhledávání; příklady.

② Ne: „navázání na formály“; komprimace (známo z organizace souborů); SQL; implementační detaily; příliš obecný a nezáživný výklad; teorie.

③ Velká variabilita zkušeností, názorů, konstruktivnosti i kultury anketních odpovědí (zapsaných okolo sta studentů od prvního do šestého ročníku studia).

④ Domácí úkoly ⟶ vnitrosemestrová písemka.

Morris-Prattův algoritmus (MP)

Idea: Ztráty naivního algoritmu vznikají tím, že v případě neshody vzorku s textem se vzorek posune o jednu pozici doprava a znovu se celý kontroluje. To znehodnocuje informaci, která byla získána předchozími porovnáními znaků. Snahou je se v případě neshody posunout se vzorkem doprava tak, aby nebylo nutné se ve vlastním textu vracet.

Hlavní část algoritmu (K)MP

```
var text: array[1..T] of char; vzorek: array[1..V] of char;
i, j: integer; nalezen: boolean;
i := 1;                                          ▷ index do textu
j := 1;                                          ▷ index do vzorku
while (i ≤ T) and (j ≤ V) do
    while (j > 0) and (text[i] ≠ vzorek[j]) do
        j := h[j];
    end while
    i := i + 1; j := j + 1
end while
nalezen := j > V;                    ▷ pokud nalezen, je na pozici i − V − 1
```

Analýza (K)MP

- Složitost $O(T)$ plus složitost předzpracování (vytvoření pole h).
- Animace trasování hlavní části KMP.

Knuth-Morris-Prattův algoritmus

- *h* se uplatní, když předpona vzorku $v_1 v_2 \ldots v_{j-1}$ je srovnána s podřetězcem textu $t_{i-j+1} t_{i-j+2} \ldots t_{i-1}$ a $v_j \neq t_i$
- mohu skočit o víc než 1? o *j*? jak *h* spočtu?
- $h(j)$ největší $k < j$ takové, že $v_1 v_2 \ldots v_{k-1}$ je příponou $v_1 v_2 \ldots v_{j-1}$, tj. $v_1 v_2 \ldots v_{k-1} = v_{j-k+1} v_{j-k+2} \ldots v_{j-1}$ a $v_j \neq v_k$
- KMP: zpětné přechody tak dlouho, až $j = 0$ (předpona vzorku není v prohledávaném textu obsažena) nebo $t_i = v_j$ ($v_1 v_2 \ldots v_j = t_{i-j+1} t_{i-j+2} \ldots t_{i-1} t_i$)
- animace lecroq, dále [POK, strana 27], též viz podklady [MAR] pro detailní popis.

Konstrukce *h* pro KMP

```
i:=1; j:=0; h[1]:=0;
while (i<V) do
  begin while (j>0) and (v[i]<>v[j]) do j:=h[j];
    i:=i+1; j:=j+1;
    if (i<=V) and (v[i]=v[j])
    then h[i]:=h[j] else h[i]:=j (*MP*)
  end;
```

Složitost konstrukce, t. j. předzpracování, je $O(V)$, celkem tedy $O(T + V)$.

Příklad: *h* pro *ababa*. KMP vs. MP.

Univerzální vyhledávací algoritmus,

který používá tabulku přechodů g odvozenou z hledaného vzorku.
(g odpovídá přechodové funkci δ KA):

```
var i,T:integer; nalezen: boolean;
text: array[1..T] of char; state,q0: TSTATE;
g:array[1..maxstate,1..maxsymb] of TSTATE;
F: set of TSTATE;...
begin
  nalezen:= FALSE; state:= q0; i:=0;
  while (i <= T) and not nalezen do
    begin
      i:=i+1; state:= g[state,text[i]];
      nalezen:= state in F;
    end;
  end;
```

Jak předzpracovat vzorek do g?

Vyhledávací stroj

Vyhledávací stroj (VS) pro sousměrné vyhledávání

- **VS pro sousměrné vyhledávání** $A = (Q, T, g, h, q_0, F)$
  - Q konečná množina stavů.
  - T konečná vstupní abeceda.
  - $g: Q \times T \to Q \cup \{\text{fail}\}$ dopředná přechodová fce.
  - $h: (Q - q_0) \to Q$ zpětná přechodová fce.
  - q_0 počáteční stav.
  - F množina koncových stavů.
- **hloubka stavu q**: $d(q) \in N_0$ je délka nejkratší dopředné posloupnosti přechodů z q_0 do q.

Vyhledávací stroj (pokr.)

- Vlastnosti g, h:
  - $g(q_0, a) \neq \text{fail}$ pro $\forall a \in T$ (neprovádí se žádný zpětný přechod v počátečním stavu).
  - je-li $h(q) = p$, pak $d(p) < d(q)$ (počet zpětných přechodů bude shora omezen součinem maximální hloubky stavu c a celkového počtu dopředných přechodů V). Rychlost vyhledávání tedy bude lineární vzhledem k V.

Konfigurace, přechod VS

- ***konfigurace VS*** (q, w), $q \in Q$, $w \in T^*$ dosud neprohledaná část textu.
- ***počáteční konfigurace VS*** (q_0, w), w je celý prohledávaný text.
- ***akceptující konfigurace VS*** (q, w), $q \in F$, w je dosud neprohledávaný text, nalezený vzorek je bezprostředně před w.
- ***přechod VS***: relace $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
  - $g(q, a) = p$, pak $(q, aw) \vdash (p, w)$ ***dopředný přechod*** pro $\forall w \in T^*$.
  - $h(q) = p$, pak $(q, w) \vdash (p, w)$ ***zpětný přechod*** pro $\forall w \in T^*$.

Vyhledávání pomocí VS

Při dopředném přechodu je přečten jeden vstupní symbol a stroj přechází do následujícího stavu p. Je-li však $g(q, a) = \text{fail}$, provede se zpětný přechod bez čtení vstupního symbolu.

Časová složitost

$S = O(T)$ (měříme počet přechodů VS).

Konstrukce VS pro KMP pro vzorek $v_1 v_2 \dots v_V$

1. počáteční stav q_0.
2. $g(q, v_{j+1}) = q'$, kde q' odpovídá předponě $v_1 v_2 \dots v_j v_{j+1}$.
3. pro q_0 definujme $g(q_0, a) = q_0$ pro $\forall a$, pro která $g(q_0, a)$ nebylo definováno v předchozím kroku.
4. $g(q, a) = \underset{\sim}{\text{fail}}$ pro $\forall q$ a a, pro která $g(q, a)$ nebylo definováno v předchozích krocích.
5. stav odpovídající úplnému vzorku je koncový.
6. zpětnou přechodovou fci h definuje na straně 47 uvedený algoritmus.

Karp-Rabinovo vyhledávání

Zcela jiný přístup: použití hashovací funkce. Místo přikládání vzoru k textu na všech pozicích kontrolujme shodu jen tam, kde podřetězec textu vypadá „podobně“. Podobnost určuje hashovací funkce. Ta by měla být

- efektivně vyčíslitelná,
- a měla by dobře separovat různé řetězce.

Vyhledávání KR je v nejhorším případě kvadratické, ale průměrně $O(T + V)$.

Karp-Rabinovo vyhledávání (pokr.)

```
#define REHASH(a, b, h) (((h-a*d)<<1+b)
void KR(char *y, char *x, int n, int m) {
int hy, hx, d, i;
/* predzpracovani: vypocet d = 2^(m-1) */
d=1; for (i=1; i<m; i++) d<<=1;
hx=hy=0;
for (i=0; i<m; i++)
  { hx=((hx<<1)+x[i]); hy=((hy<<1)+y[i]); }
/* vyhledavani */
for (i=m; i<=n; i++) {
  if (hy==hx) && strncmp(y+i-m,x,m)==0) OUTPUT(i-m);
  hy=REHASH(y[i-m], y[i], hy);
} }
```

Karp-Rabinovo vyhledávání (pokr.)

Příklad: ([HCS, Ch. 6]) $V = ing$, $T = string\ matching$.

Předzpracování: $hash = 105 \times 2^2 + 110 \times 2 + 103 = 743$.

Vyhledávání:

T=	s	t	r	**i**	**n**	**g**	
hash=			806	797	776	**743**	678

m	a	t	c	h	**i**	**n**	**g**
585	443	746	719	766	709	736	**743**

Část V

Vyhledávání (konečné) množiny vzorků

Vyhledávání množiny vzorků

VS pro sousměrné vyhledávání množiny vzorků $p = \{v^1, v^2, \ldots, v^P\}$

Místo opakovaného procházení textu pro každý vzorek jen „jeden“ průchod (KA).

Obecný algoritmus VS

```
var text: array[1..T] of char;
  i: integer; nalezen: boolean; state: tstate;
  g: array[1..maxstate,1..maxsymbol] of tstate;
  h: array[1..maxstate] of tstate; F: set of tstate;
nalezen:=false; state:=q0; i:=0;
while (i<=T) and not nalezen do
begin i:=i+1;
  while g[state,text[i]]=fail do state:=h[state];
  state:=g[state,text[i]]; nalezen:=state in F
end
```

Obecný algoritmus VS (pokr.)

- Konstrukce přechodových funkcí h, g?
- Jak pro P vzorků? Hlavní myšlenka?
- Aho, Corasicková, 1975 (AC vyhledávací stroj).

Algoritmus Aho a Corasickové I

Konstrukce g pro AC VS pro množinu vzorků $p = \{v^1, v^2, \ldots, v^P\}$

1. Počáteční stav q_0.
2. $g(q, b_{j+1}) = q'$, kde q' odpovídá předponě $b_1 b_2 \ldots b_{j+1}$ vzorku v^i, pro $\forall i \in \{1, \ldots, P\}$.
3. Pro q_0 definujme $g(q_0, a) = q_0$ pro $\forall a$, pro která $g(q_0, a)$ nebylo definováno v předchozím kroku.
4. $g(q, a) = \underset{\sim}{\text{fail}}$ pro $\forall q$ a a, pro která $g(q, a)$ nebylo definováno v předchozích krocích.
5. Stav odpovídající úplnému vzorku je koncový.

Příklad: $p = \{\text{he, she, her}\}$ nad $T = \{\text{h, e, r, s, x}\}$, kde x je cokoliv jiného než $\{\text{h, e, r, s}\}$.

Chybová fce h (AC II)

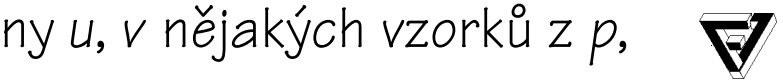
Konstrukce h pro AC VS pro množinu vzorků $p = \{v^1, v^2, \ldots, v^P\}$

Nejprve definujeme chybovou funkci f induktivně vzhledem k hloubce stavů takto:

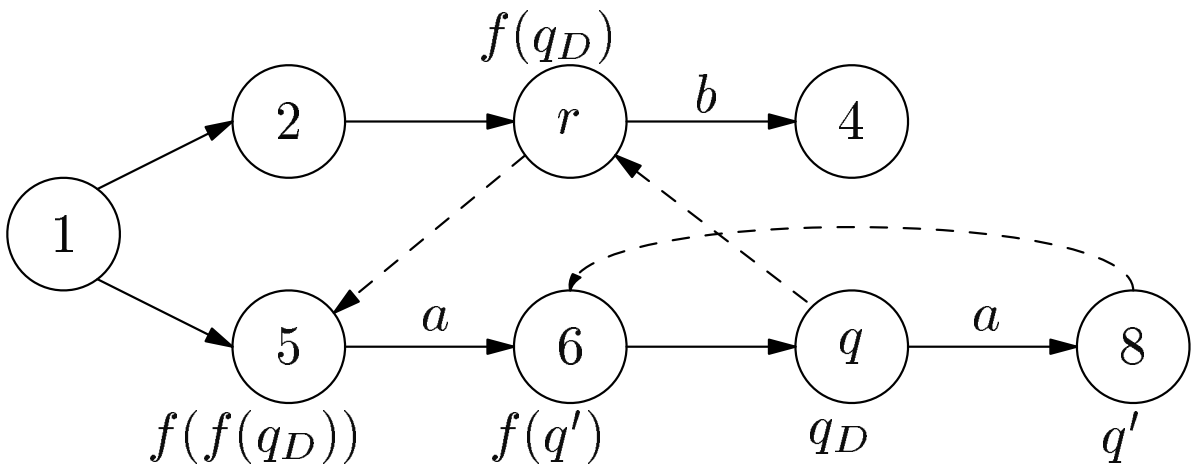
1. Pro $\forall q$ hloubky 1 bude $f(q) = q_0$.
2. Předpokládejme, že f je definována pro všechny stavy hloubky d a menší. Buď q_D stav hloubky d a $g(q_D, a) = q'$. Pak $f(q')$ spočítáme takto:

   $q := f(q_D)$;
   while $g(q, a) =$ fail **do** $q := f(q)$;
   $f(q') := g(q, a)$.

- Cyklus skončí, neboť $g(q_0, a) \neq$ fail.
- Jestliže stavy q, r reprezentují předpony u, v nějakých vzorků z p, pak $f(q) = r \Leftrightarrow v$ je nejdelší vlastní přípona u.



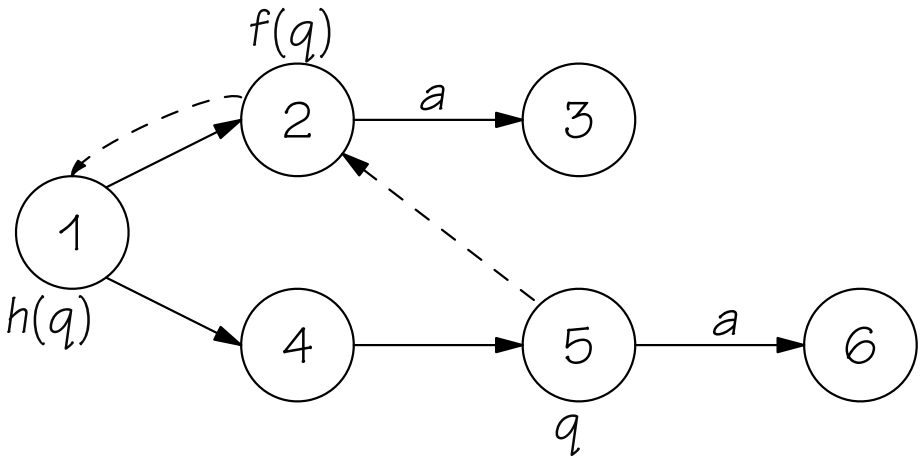
Chybová fce h (AC III)



Konstrukce h pro AC VS pro množinu vzorků $p = \{v^1, v^2, \ldots, v^P\}$ (pokračování)

- Za zpětnou přechodovou fci h můžeme vzít f, mohou se však provádět zbytečné zpětné přechody.
- Funkci h definujeme induktivně vzhledem k hloubce stavů takto:
  - Pro $\forall$ stav q hloubky 1 bude $h(q) = q_0$.
  - Předpokládejme, že h je definována pro všechny stavy hloubky d a menší. Nechť hloubka q je $d + 1$. Jestliže množina znaků, pro které je ve stavu $f(q)$ hodnota funkce g jiná hodnota než fail, je podmnožinou množiny znaků, pro které je hodnota funkce g ve stavu q jiná než fail, pak $h(q) := h(f(q))$, jinak $h(q) := f(q)$.

Konstrukce h pro AC VS (pokr.)



Vyhledávací konečné automaty

Deterministický konečný automat (DKA) $M=(K,T,\delta,q_0,F)$

1. K je konečná množina vnitřních stavů.
2. T je konečná vstupní abeceda.
3. δ je zobrazení z $K \times T$ do K.
4. $q_0 \in K$ je počáteční stav.
5. $F \subseteq K$ je množina koncových stavů.

Vyhledávací konečné automaty

1. ***Úplně určený automat*** jestliže δ definována pro každou dvojici $(q, a) \in K \times T$, jinak ***neúplně určený automat***.
2. ***Konfigurace M*** je dvojice (q, w), kde $q \in K$, $w \in T^*$ je dosud neprohledaná část textu.
3. ***Počáteční konfigurace M*** je (q_0, w), kde w je celý prohledávaný text.
4. ***Koncová konfigurace M*** je (q, w), kde $q \in F$ a $w \in T^*$.

Vyhledávání pomocí KA M

Při přechodu je přečten jeden vstupní symbol a stroj přechází do následujícího stavu p.

- ***Přechod M***: dán stavem a symbolem na vstupu; relace $\vdash \subseteq (K \times T^*) \times (K \times T^*)$; jestliže $\delta(q, a) = p$, pak $(q, aw) \vdash (p, w)$ pro každé $\forall w \in T^*$.
- ***k-tá mocnina***, ***tranzitivní*** resp. ***tranzitivně-reflexivní uzávěr*** relace $\vdash$: $\vdash^k$, $\vdash^+$, $\vdash^*$.
- $L(M) = \{w \in T^* : (q_0, w) \vdash^* (q, w')$ pro nějaké $q \in F, w' \in T^*\}$ ***jazyk přijímaný KA*** M.
- časová složitost $O(T)$ (měříme počet přechodů KA M).

Nedeterministický KA

Definice: ***Nedeterministický konečný automat*** (NKA) je $M = (K, T, \delta, q_0, F)$, kde K, T, q_0, F jsou stejné jako u deterministické verze KA, ale
$\delta : K \times T \rightarrow 2^K$
$\delta(q, a)$ je nyní ***množina*** stavů.

Definice: $\vdash \in (K \times T^*) \times (K \times T^*)$ ***přechod***: jestliže $p \in \delta(q, a)$, pak $(q, aw) \vdash (p, w)$ pro $\forall w \in T^*$.

Definice: Přijetí řetězce, $L(M)$ analogicky jako u DKA.

Konstrukce VS (DKA) z NKA

Věta: Pro každý nedeterministický konečný automat $M=(K,T,\delta,q_0,F)$ můžeme sestrojit deterministický konečný automat $M'=(K',T,\delta',q'_0,F')$ takový, že $L(M) = L(M')$.

Konstrukce VS (DKA) z NKA (pokr.)

Konstruktivní důkaz (algoritmus):
Vstup: Nedeterministický KA $M = (K, T, \delta, q_0, F)$
Výstup: Deterministický KA

① $K' = \{\{q_0\}\}$, stav $\{q_0\}$ neoznačený.

② Jestliže v K' jsou všechny stavy označeny, pokračuj krokem 4.

③ Vybereme z K' neoznačený stav q':

- $\delta'(q', a) = \bigcup\{\delta(p, a)\}$ pro $\forall p \in q'$ a $a \in T$;
- $K' = K' \cup \delta'(q', a)$ pro $\forall a \in T$;
- označíme q' a pokračujeme krokem 2.

④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Konstrukce g pro VS

Konstrukce g' pro VS pro množinu vzorků $p = \{v^1, v^2, \ldots, v^P\}$

① Vytvoříme NKA M:

- Počáteční stav q_0.
- Pro $\forall a \in T$ definujme $g(q_0, a) = q_0$.
- Pro $\forall i \in \{1, \ldots, P\}$ definujme $g(q, b_{j+1}) = q'$, kde q' odpovídá předponě $b_1 b_2 \ldots b_{j+1}$ vzorku v^i.
- Stav odpovídající úplnému vzorku je koncový.

② a jemu odpovídající DKA M' s g'.

Osnova (Týden třetí)

1. Opakování: animace KMP, AC.
2. Algoritmy pro sousměrné vyhledávání nekonečné množiny vzorků.
3. Regulární výrazy.
4. Přímá konstrukce (N)KA pro daný RV.

Část VI

Vyhledávání nekonečné množiny vzorků

Regulární výraz (RV)

Definice: ***Regulární výraz V nad abecedou A***:

1. $\varepsilon, \boldsymbol{O}$ jsou RV a pro $\forall a \in A$ je a RV.
2. Jestliže x, y RV nad A, pak:
   - $(x + y)$ je RV (sjednocení);
   - $(x.y)$ je RV (zřetězení);
   - $(x)^*$ je RV (iterace).

Konvence o prioritě regulárních operací:
sjednocení < zřetězení < iterace.
Definice: Pak za ***(zobecněný) regulární výraz*** považujeme i takové termy, které neobsahují vzhledem k této konvenci nepotřebné závorky.

Hodnota RV

Hodnota h(x) RV x**:**

① $h(\mathbf{O}) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$

②
- $h(x + y) = h(x) \cup h(y)$
- $h(x.y) = h(x).h(y)$
- $h(x^*) = (h(x))^*$

☞ $h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \ldots$

☞ Hodnotou RV je regulární jazyk (RJ).

☞ Každý RJ lze reprezentovat RV.

☞ Pro $\forall$ RV V $\exists$ KA M: $h(V) = L(M)$.

Axiomatizace RV (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ asociativnost sjednocení

A2: $x.(y.z) = (x.y).z = x.y.z$ asociativnost zřetězení

A3: $x + y = y + x$ komutativnost sjednocení

A4: $(x + y).z = x.z + y.z$ distributivnost zprava

A5: $x.(y + z) = x.y + x.z$ distributivnost zleva

A6: $x + x = x$ idempotence sjednocení

A7: $\varepsilon.x = x$ jednotkový prvek pro zřetězení

A8: $\mathbf{0}.x = \mathbf{0}$ nulový prvek pro zřetězení

A9: $x + \mathbf{0} = x$ jednotkový prvek pro sjednocení

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$

Podobnost regulárních výrazů

Věta: Tato axiomatizace RV je úplná a bezesporná.

Definice: Regulární výrazy nazveme ***podobné***, jestliže se mezi sebou dají navzávem převést pomocí axiomů A1 až A11.

Věta: Podobné regulární výrazy mají stejnou hodnotu.

Délka regulárního výrazu

Definice: ***Délka d(V) regulárního výrazu V***:

1. Je-li V tvořen jedním symbolem, pak $d(V) = 1$.
2. $d(V_1 + V_2) = d(V_1) + d(V_2) + 1$.
3. $d(V_1.V_2) = d(V_1) + d(V_2) + 1$.
4. $d(V^*) = d(V) + 1$.
5. $d((V)) = d(V) + 2$.

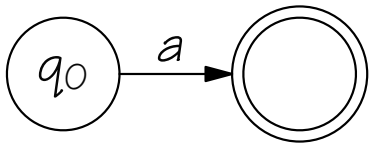
Konstrukce NKA pro daný RV

Definice: ***Zobecněný NKA*** *dovoluje ε-přechody (přechody bez čtení vstupního symbolu).*

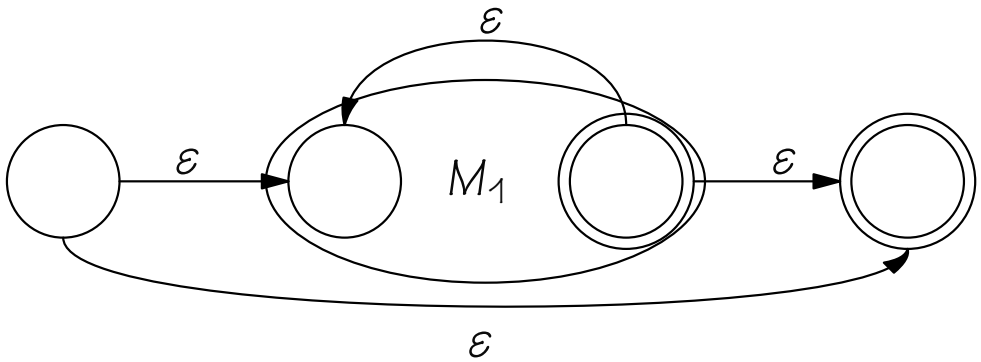
Věta: Pro každý RV V je možno sestrojit KA M takový, že $h(V) = L(M)$.
Důkaz: Strukturní indukcí vzhledem k RV V:

Konstrukce NKA pro daný RV (důkaz)

① $V = a$

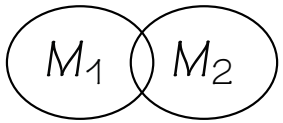


② $V = V_1^*$ M_1 automat pro V_1 $(h(V_1) = L(M_1))$

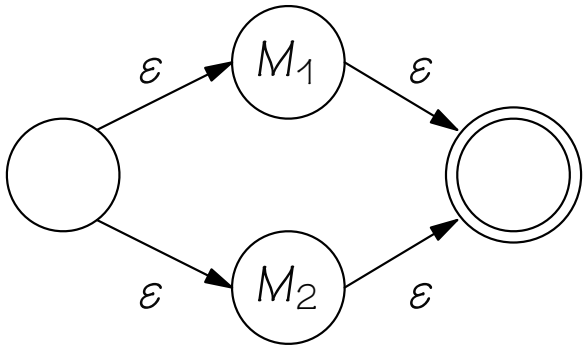


Konstrukce NKA pro daný RV (důkaz pokr.)

③ $V = V_1 \cdot V_2$



④ $V = V_1 + V_2$ M_1, M_2 automaty pro V_1, V_2 ($h(V_1) = L(M_1)$,



$h(V_2) = L(M_2)$)

Konstrukce NKA pro daný RV (pokr.)

Vlastnosti takto vytvořeného NKA:

- Z každého stavu vycházejí maximálně 2 hrany.
- Z koncových stavů nevycházejí žádné hrany.
- Počet stavů $M \leq 2 \cdot d(V)$.
- Simulace chování automatu M v čase $O(d(V)T)$ a paměti $O(d(V))$.

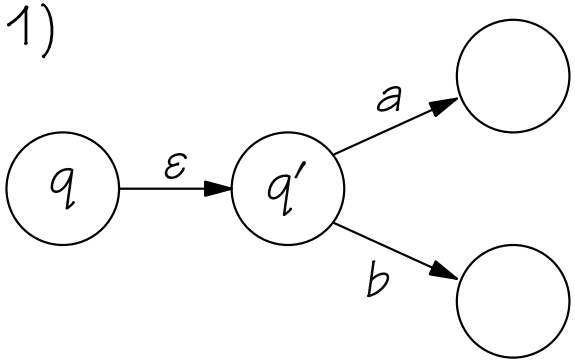
Simulace NKA

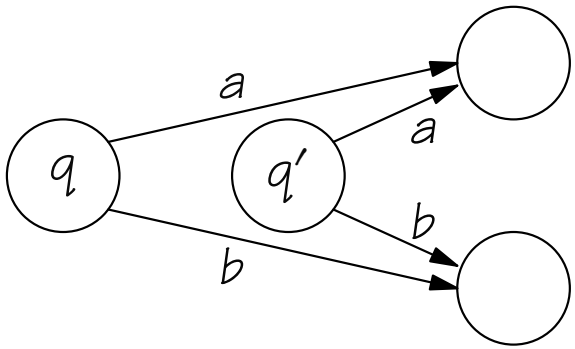
Eliminace ε-přechodů

Pro následující metody simulace NKA je třeba odstranit ε-přechody.

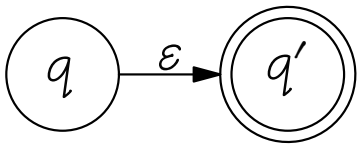
Toto můžeme provést známým postupem:

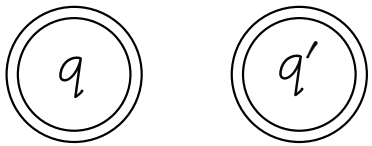
1)





2)





Simulace NKA (pokr.)

Metody implementace simulace NKA

Stav reprezentujeme booleovským vektorem a současně budeme procházet všemi možnými cestami. Dvě možnosti:

- Obecný program s použitím tabulky přechodů.
- Implementace automatu ve formě (generovaného) programu pro konkrétní automat.

Přímá konstrukce (N)KA pro daný RV

Nechť V je RV nad abecedou T. Pak KA $M = (K, T, \delta, q_0, F)$ takový, že $h(V) = L(M)$ sestrojíme takto:

1. Očíslujeme všechny výskyty symbolů z T ve výrazu V různými přirozenými čísly. Dostaneme V'.
2. Množina počátečních symbolů $Z = \{x_i : \text{symbolem } x_i \text{ může začínat nějaký řetězec z } h(V'), x_i \neq \varepsilon\}$.
3. Množina sousedů $P = \{x_i y_j : \text{symboly } x_i \neq \varepsilon \neq y_j \text{ mohou být vedle sebe v nějakém řetězci z } h(V')\}$.
4. Množina koncových symbolů $F = \{x_i : \text{symbolem } x_i \neq \varepsilon \text{ může končit nějaké slovo z } h(V')\}$.
5. Množina stavů $K = \{q_0\} \cup Z \cup \{y_j : x_i y_j \in P\}$.
6. Přechodová funkce δ:
   - $\delta(q_0, x)$ obsahuje x_i pro $\forall x_i \in Z$ vzniklá očíslováním x.
   - $\delta(x_i, y)$ obsahuje y_j pro $\forall x_i y_j \in P$ takové, že y_j vzniklo očíslováním y.
7. F je množina koncových stavů.

Přímá konstrukce (N)KA pro daný RV

Příklad 1: $R = ab^*a + ac + b^*ab^*$.

Příklad 2: $R = ab* + ac + b^*a$.

Osnova (Týden čtvrtý)

① Vlastnosti regulárních výrazů.

② Přímá konstrukce DKA ekvivalentního konečného automatu pro daný RV pomocí derivace RV.

③ Derivace regulárních výrazů pozičním vektorem.

④ Protisměrné vyhledávání (BMH, CW, BUC).

⑤ Proximitní vyhledávání.

Derivace regulárního výrazu

Definice: **Derivace $\frac{dV}{dx}$ regulárního výrazu V podle řetězce $x \in T^*$**:

① $\frac{dV}{d\varepsilon} = V.$

② Pro $a \in T$ platí:

$$\frac{d\varepsilon}{da} = \emptyset$$

$$\frac{db}{da} = \begin{cases} \emptyset & \text{jestliže } a \neq b \\ \varepsilon & \text{jestliže } a = b \end{cases}$$

$$\frac{d(U+V)}{da} = \frac{dU}{da} + \frac{dV}{da}$$

$$\frac{d(U.V)}{da} = \begin{cases} \frac{dU}{da} \cdot V + \frac{dV}{da} & \text{jestliže } \varepsilon \in h(U) \\ \frac{dU}{da} \cdot V & \text{jinak} \end{cases}$$

$$\frac{d(V^*)}{da} = \frac{dV}{da} \cdot V^*$$

Derivace regulárního výrazu (pokr.)

③ Pro $x = a_1 a_2 \dots a_n$, $a_i \in T$ platí

$$\frac{dV}{dx} = \frac{d}{da_n}\left(\frac{d}{da_{n-1}}\left(\cdots \frac{d}{da_2}\left(\frac{dV}{da_1}\right)\cdots\right)\right).$$

Vlastnosti regulárních výrazů

Příklad: Derivujte $V = fi + fi^* + f^*ifi$ podle i a f.
Příklad: Derivujte $(o^*sle)^*cno$ podle o, s, l, c a $osle$.

Věta: $h\left(\frac{dV}{dx}\right) = \{y : xy \in h(V)\}$.
Příklad: Dokažte výše uvedenou větu. Návod: strukturní indukcí vzhledem k V a x.

Definice: ***Regulární výrazy x, y jsou podobné*** jestliže jeden z nich může být transformován na druhý pomocí axiomů axiomatické teorie RV (Salomaa).

Příklad: Existuje RV podobný $V = fi + fi^* + f^*ifi$ délek 7, 15?

Přímá konstrukce DKA pro daný RV (pomocí derivací RV)

Brzozowski (1964, Journal of the ACM)

Vstup: RV V nad T.

Výstup: KA $M = (K, T, \delta, q_0, F)$ takový, že $h(V) = L(M)$.

1. Položme $Q = \{V\}$, $Q_0 = \{V\}$, $i := 1$.
2. Vytvořme derivace všech výrazů z Q_{i-1} podle všech symbolů z T. Do Q_i vložíme všechny výrazy vzniklé derivací výrazů z Q_{i-1}, které nejsou podobné výrazům z Q.
3. Jestliže $Q_i \neq \emptyset$, přidáme Q_i do Q, položíme $i := i + 1$ a přejdeme na krok 2.
4. Pro $\forall \frac{dU}{dx} \in Q$ a $a \in T$ položíme $\delta\left(\frac{dU}{dx}, a\right) = \frac{dU}{dx'}$, v případě, že výraz $\frac{dU}{dx'}$ je podobný výrazu $\frac{dU}{d\,xa}$. (Přitom $\frac{dU}{dx'} \in Q$.)
5. Množina $F = \left\{\frac{dU}{dx} \in Q : \varepsilon \in h\left(\frac{dU}{dx}\right)\right\}$.

Příklad: RV= $R = (0 + 1)^*1$.

$Q = Q_0 = \{(0 + 1)^*1\}, i = 1$

$Q_1 = \{\frac{dR}{d0} = R, \frac{dR}{1}\} = \{(0 + 1)^*1 + \varepsilon\}$

$Q_2 = \{\frac{(0+1)^*1+\varepsilon}{d0} = R, \frac{(0+1)^*1+\varepsilon}{d1} = (0 + 1)^*1 + \varepsilon\} = \emptyset$

Příklad: RV= $(10)^*(00)^*1$.

Více viz Watson, B. W.: *A taxonomy of finite automata construction algorithms*, Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993.
citeseer.ist.psu.edu/watson94taxonomy.html

Derivace RV pozičním vektorem I

Definice: Poziční vektor je množina čísel, které odpovídají pozicím takových symbolů abecedy, které se mohou vyskytnout na začátku zbytku řetězu, který je součástí hodnoty daného RV.

Příklad: Mějme regulární výraz:

$$a \quad . \quad b^* \quad . \quad c \tag{1}$$

K označení pozice budeme používat zobáček $\wedge$. Na začátku bude tedy výraz **(1)** označen takto:

$$\hat{a} \quad . \quad b^* \quad . \quad c \tag{2}$$

Derivací označeného regulárního výrazu dostaneme nově označený regulární výraz. Základní pravidlo pro derivaci je toto:

1. Je-li označen operand, podle kterého se derivuje, pak se označí místa následující za tímto operandem. Jeho označení se ruší. To znamená, že derivací výrazu **(2)** podle operandu a dostaneme:

$$a \quad . \quad \overset{\wedge}{b^*} \quad . \quad c \tag{3a}$$

Derivace RV pozičním vektorem II

2. Protože je označena konstrukce, která generuje také prázdný řetězec, označíme také konstrukci následující:

$$a \;.\; \overset{\wedge}{b^*} \;.\; \hat{c} \qquad \textbf{(3b)}$$

Nyní derivací podle operandu b výrazu **(3b)** dostaneme:

$$a \;.\; b^* \;.\; \hat{c} \qquad \textbf{(4a)}$$

3. Protože je označena konstrukce následující za konstrukcí v iteraci, musí se označit i předchozí konstrukce. $a \;.\; \overset{\wedge}{b^*} \;.\; \hat{c}$ **(4b)**

Derivací výrazu **(4b)** podle operandu c dostaneme:

$$a \;.\; b^* \;.\; \underset{\wedge}{c} \qquad \textbf{(5)}$$

Takto označený regulární výraz odpovídá prázdnému regulárnímu výrazu ε.

Derivace RV pozičním vektorem III

Shrnutí postupu derivování:

- Ke každé syntaktické konstrukci se vytvoří seznam počátečních pozic na začátcích členů.
- Je-li symbol v konstrukci roven symbolu, podle kterého se provádí derivace, a je-li na označené pozici, pak se označení posouvá před následující pozici.
- Je-li za konstrukcí operátor iterace a označení je na konci konstrukce, pak se do výsledného seznamu připojí také seznam počátečních pozic náležející této konstrukci.
- Je-li označení před nějakou konstrukcí, pak se do výsledného seznamu připojí seznam počátečních pozic této konstrukce.
- Je-li označení před konstrukcí, která generuje také prázdný řetěz, pak se do výsledného seznamu připojí také seznam počátečních pozic konstrukce následující.
- Má-li se označit konstrukce v závorce, pak je třeba označit začátky všech členů v závorce.

Derivace RV pozičním vektorem: příklad

Příklad: $a.b^*.c$, derivace podle a, b, c.

Část VII

Protisměrné vyhledávání

Protisměrné vyhledávání

Protisměrné vyhledávání – princip. Může být směr podstatný?
V jakých případech?

Přehled vyhledávacích algoritmů:

- 1 vzorek – Boyer-Moore (BM, 1977), Boyer-Moore-Horspool (BMH, 1980), Boyer-Moore-Horspool-Sunday (BMHS, 1990).
- n vzorků – Commentz-Walterová (CW, 1979).
- nekonečná množina vzorků: reverzovaný regulární výraz – Bucziłowski (BUC).

Boyer-Moore-Horspool algoritmus

```
 1: var: TEXT: array[1..T] of char;
 2:     VZOREK: array[1..V] of char; I,J: integer; NALEZEN: boolean;
 3: NALEZEN := false;    I := V;
 4: while (I ≤ T) and not NALEZEN do
 5:         J := 0;
 6:         while (J < V) and (VZOREK[V − J] = TEXT[I − J]) do
 7:                 J := J + 1;
 8:         end while
 9:         NALEZEN := (J = V);
10:
11:         if not NALEZEN then
12:                 I := I + SHIFT(TEXT[I − J], J)
13:         end if
14: end while
```

SHIFT(A, J) = **if** A se nevyskytuje v ještě nesrovnalé části vzorku **then** V − J **else** nejmenší K takové, že VZOREK[V − (J + K)] = A;

Kdy rychlejší než KMP? Kdy $O(T/V)$?

Příklad: Hledání vzorku **BANANA** v textu
I-WANT-TO-FLAVOR-NATURAL-BANANAS.

CW algoritmus

Idea: AC + protisměrnost (BM) [1979]

```
const LMIN=/délka nejkratšího vzorku/
var TEXT: array [1..T] of char; I, J: integer;
    NALEZEN: boolean;  STATE: TSTATE;
    g: array [1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F: set of TSTATE;
begin
 NALEZEN:=FALSE; STATE:=q0; I:=LMIN; J:=0;
 while (I<=T) & not (NALEZEN) do
  begin
   if g[STATE, TEXT[I-J]]=fail
    then begin I:=I+SHIFT[STATE, TEXT[I-J]];
               STATE:=q0; J:=0;
         end
    else begin STATE:=g[STATE, TEXT[I-J]]; J:=J+1 end
   NALEZEN:=STATE in F
  end
end
```

Konstrukce CW vyhledávacího stroje

VSTUP: Množina vzorků $P = \{v_1, v_2, \ldots, v_k\}$
VÝSTUP: CW vyhledávací stroj
METODA: Sestrojíme funkci g a zavedeme ohodnocení w jednotlivých stavů:

1. Počáteční stav q_0; $w(q_0) = \varepsilon$.
2. Každý stav vyhledávacího stroje odpovídá příponě $b_m b_{m+1} \ldots b_n$ nějakého vzorku v_i z množiny p. Definujeme $g(q, b_{m-1}) = q'$, kde q' odpovídá příponě $b_{m-1} b_m b_{m+1} \ldots b_n$ vzorku v_i:
   $w(q) = b_n \ldots b_{m+1} b_m$; $w(q') = w(q) b_{m-1}$.
3. $g(q, a) = \underset{\sim}{\text{fail}}$ pro všechna q a a, pro která $g(q, a)$ nebylo definováno v kroku 2.
4. Každý stav, který odpovídá úplnému vzorku, bude koncový.

CW – funkce *shift*

Definice: $shift[STATE, TEXT[l - J]] = \min\{A, shift2(STATE)\}$, kde $A = \max\{shift1(STATE), char(TEXT[l - J]) - J - 1\}$.

Jednotlivé funkce jsou definovány takto:

1. $char(a)$ je definována pro všechny symboly z abecedy T jako nejmenší hloubka stavu, do kterého se v CW vyhledávacím stroji přechází při symbolu a. Pokud symbol a není v žádném vzorku, je $char(a) = \text{LMIN} + 1$, kde LMIN je délka nejkratšího vzorku. Formálně:
   $char(a) = \min\{\text{LMIN} + 1, \min\{d(q) : w(q) = ax, x \in T^*\}\}$.
2. Funkce $shift1(q_0) = 1$, pro ostatní stavy má hodnotu $shift1(q) = \min\{\text{LMIN}, A\}$, kde $A = \min\{k : k = d(q') - d(q), w(q') = w(q)u$ pro nějaké neprázdné slovo $u\}$. Stav q' má větší hloubku než q a $w(q)$ je vlastní předponou $w(q')$.
3. Funkce $shift2(q_0) = \text{LMIN}$, pro ostatní stavy má hodnotu $shift2(q) = \min\{A, B\}$, kde $A = \min\{k : k = d(q') - d(q), w(q') = w(q)u$ pro nějaké neprázdné slovo u, q' je koncový stav$\}$, $B = shift2(q') : q'$ je předchůdce q.

CW – funkce *shift*

Příklad: $P = \{cacbaa, aba, acb, acbab, ccbab\}$.

LMIN = 3,

char	a	b	c	X
	1	1	2	4

stav	shift1	shift2
q_0	1	3
a	1	2
b	1	3
aa	3	2
ab	1	2
bc	2	3
ba	1	1
aab	3	2
aba	3	2
bca	2	2
bab	3	1
aabc	3	2
babc	3	1
aabca	3	2
babca	3	1
babcc	3	1
aabcac	3	2

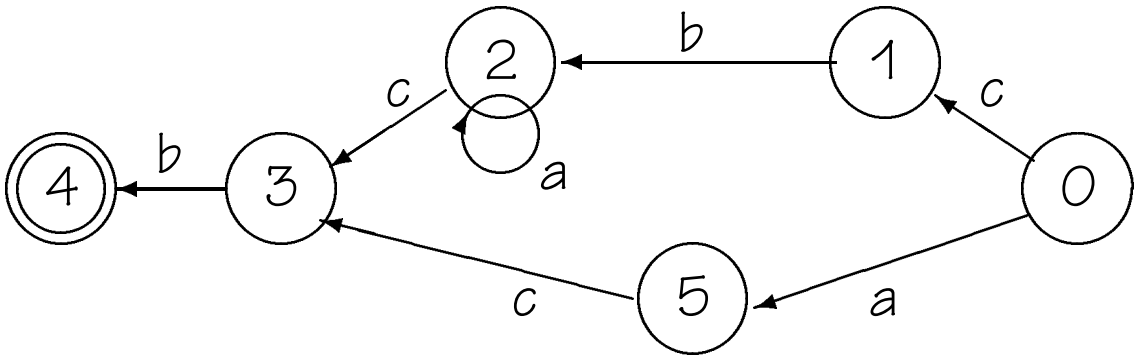
Část VIII

Vyhledávání nekonečné množiny vzorků

Protisměrné vyhl. nek. mn. vzorků

Definice: ***Reverzovaný regulární výraz*** vznikne reverzí všech zřetězení ve výrazu.

Příklad: Reverzovaný RV k $V = bc(a + a^*bc)$ je $V^R = (a + cba^*)cb$:



Protisměrné vyhl. nek. mn. vzorků (pokr.)

Buczilowski: *V* vyhledáváme tak, že vytvoříme V^R a pro každý stav a nedefinovaný přechod z něho se určí *shift*[*STAV*, *SYMBOL*] analogicky jako u algoritmu CW:

	a	b	c	X	
0		1		3	·
1	1		1	2	(3!) ·
2		1			
3	1		1	1	
4	1	1	1	1	
5	1	1		1	
	·	·		·	

Cvičení

Příklad : Mějme množinu vzorků P= {tis, ti, iti}:

- Vytvořte NKA pro vyhledávání P.
- Vytvořte DKA příslušný tomuto NKA a zminimalizujte jej. Nakreslete přechodové diagramy obou automatů (DKA a minimálního DKA) a popište postup minimalizace.
- Srovnejte s výsledkem vyhledávacího stroje AC.
- Řešte úlohu také algoritmem přímé konstrukce DKA (derivováním) a diskutujte, zda výsledkem jsou izomorfní automaty.

Dvoucestný automat se skokem I

Definice: **2DKAS** je $M = (Q, \Sigma, \delta, q_0, k, \uparrow, F)$, kde

Q množina stavů

Σ vstupní abeceda

δ zobr. $Q \times \Sigma \longrightarrow Q \times \{-1, 1, \ldots, k\}$

$q_0 \in Q$ počáteční stav

$k \in N$ max. délka skoku

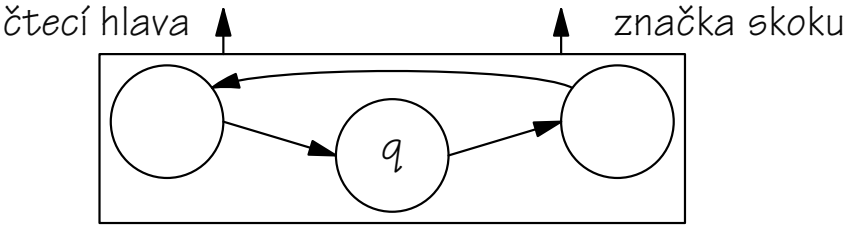
$\uparrow \notin Q \cup \Sigma$ značka skoku

$F \subseteq Q$ množina koncových stavů

Definice: **Konfigurace 2DKAS** je řetězec z $\Sigma^* Q \Sigma^* \uparrow \Sigma^*$.

Definice: **Množinu konfigurací 2DKAS M** značíme $K(M)$.

Příklad: $a_1 a_2 \ldots a_{i-1}\, q\, a_i \ldots a_{j-1} \uparrow a_j \ldots a_n \in K(M)$:



Dvoucestný automat se skokem II

Definice: **Přechod 2DKAS** je relace $\vdash \subseteq K(M) \times K(M)$ taková, že

- $a_1 \dots a_{i-1} a_i\ q\ a_{i+1} \dots a_{j-1} \uparrow a_j \dots a_n \vdash a_1 \dots a_{i-1}\ q'\ a_i a_{i+1} \dots a_{j-1} \uparrow a_j \dots a_n$ pro $i > 1, \delta(q, a_{i+1}) = (q', -1)$ (protisměrné srovnání),
- $a_1 \dots a_i\ q\ a_{i+1} \dots a_{j-1} \uparrow a_j \dots a_n \vdash a_1 \dots a_i a_{i+1} \dots a_{t-1}\ q' \uparrow a_t \dots a_n$ pro $\delta(q, a_{i+1}) = (q', m), \quad m \geq 1, \quad t = \min\{j + m, n + 1\}$ (protisměrný skok),
- $a_1 \dots a_j\ q\ a_{j+1} \dots a_{i-1} \uparrow a_i \dots a_n \vdash a_1 \dots a_j a_{j+1} \dots a_{t-1}\ q' \uparrow a_t \dots a_n$ pro $\delta(q, a_i) = (q', m), \quad m \geq 1, \quad t = \min\{i + m, n + 1\}$ (sousměrný skok), .
- $a_1 \dots a_{j-1}\ q\ a_j \dots a_{i-1} \uparrow a_i a_{i+1} \dots a_n \vdash a_1 \dots a_{j-1}\ q'\ a_j \dots a_{i-1} a_i \uparrow a_{i+1} \dots a_n$ pro $i > 1, \delta(q, a_i) = (q', 1)$ (sousměrné srovnání).

(Sousměrná pravidla jsou pro sousměrné stroje, protisměrná pro protisměrné).
Definice: $\vdash^k$, $\vdash^*$ analogicky jak u VS.

Hierarchie vyhledávacích strojů

Definice: ***Jazyk přijímaný dvoucestným automatem***
$M = (Q, \Sigma, \delta, q_0, k, \uparrow, F)$ je množina $L(M) = \{w \in \Sigma^* : q_0 \uparrow T \vdash^* w'fxw \uparrow,$ kde $f \in F, w' \in \Sigma^*, x \in \Sigma\}$.

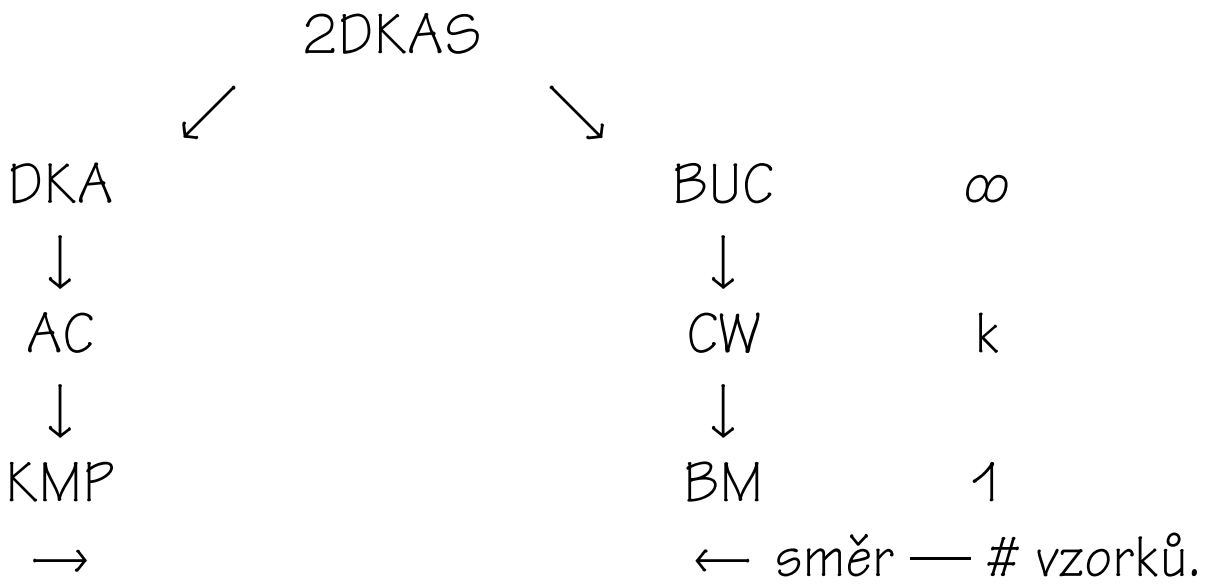
Věta: $L(M)$ pro 2DKAS M je regulární.
Příklad: Zformulujte protisměrné vyhledávání vzorku BANANA v textu I-WANT-TO-FLAVOUR-NATURAL-BANANAS pomocí BM jako 2DKAS a vyhledávání trasujte jako posloupnost konfigurací 2DKAS.

Cvičení

Mějme regulární výraz $R = 1(0 + 1^*02)$ nad abecedou $A = \{0, 1, 2\}$.

- ☞ Sestrojte DKA pro protisměrné vyhledávání R (Bucziłowski) a spočtěte chybovou funkci. Nakreslete přechodový diagram tohoto automatu včetně vizualizace chybové funkce.
- ☞ Zapište výsledný automat jako 2DKAS a trasujte vyhledávání v textu 11201012102.

Shrnutí přesného vyhledávání



Část IX

Proximitní vyhledávání

Metrika (pro proximitní vyhledávání)

Jak měřit (metrika) podobnost řetězců?

Definice: Funkci $d : S \times S \longrightarrow R$ nazvete ***metrika*** jestliže platí

1. $d(x, y) \geq 0$
2. $d(x, x) = 0$
3. $d(x, y) = d(y, x)$ (symetrie)
4. $d(x, y) = 0 \Rightarrow x = y$ (pravidlo nerozeznatelných bodů)
5. $d(x, y) + d(y, z) \geq d(x, z)$ (trojúhelníková nerovnost)

Hodnoty funkce d (distance) nazýváme ***vzdálenost***.

Metriky pro proximitní vyhledávání

Definice: Mějme řetězce X a Y nad abecedou Σ. Minimální počet editačních operací pro přeměnu X na Y je

- ***Hammingova vzdálenost***, R-vzdálenost, pokud povolíme jen operaci **R**eplace,
- ***Levenshteinova vzdálenost***, DIR-vzdálenost, pokud povolíme operace **D**elete, **I**nsert a **R**eplace,
- ***Zobecněná Levenshteinova vzdálenost***, $DIRT$-vzdálenost, pokud povolíme operace **D**elete, **I**nsert, **R**eplace a **T**ranspose. Transpozice je možná jen u sousedních znaků.

Jde o ***metriky***, Hammingova je nad řetězci stejné délky, Levenshteinovy i délek různých.

Proximitní vyhledávání – příklady

Příklad: Najděte takový příklad řetězců X a Y, že platí zároveň $R(X, Y) = 5$, $DIR(X, Y) = 5$ i $DIRT(X, Y) = 5$, nebo dokažte neexistenci takových řetězců.

Příklad: Najděte takový příklad řetězců X a Y, že platí zároveň $R(X, Y) = 5$, $DIR(X, Y) = 4$ i $DIRT(X, Y) = 3$, nebo dokažte neexistenci takových řetězců.

Příklad: Najděte takový příklad řetězců X a Y délky $2n$, $n \in N$, že $R(X, Y) = n$ a a) $DIR(X, Y) = 2$; b) $DIRT(X, Y) = \lceil \frac{n}{2} \rceil$

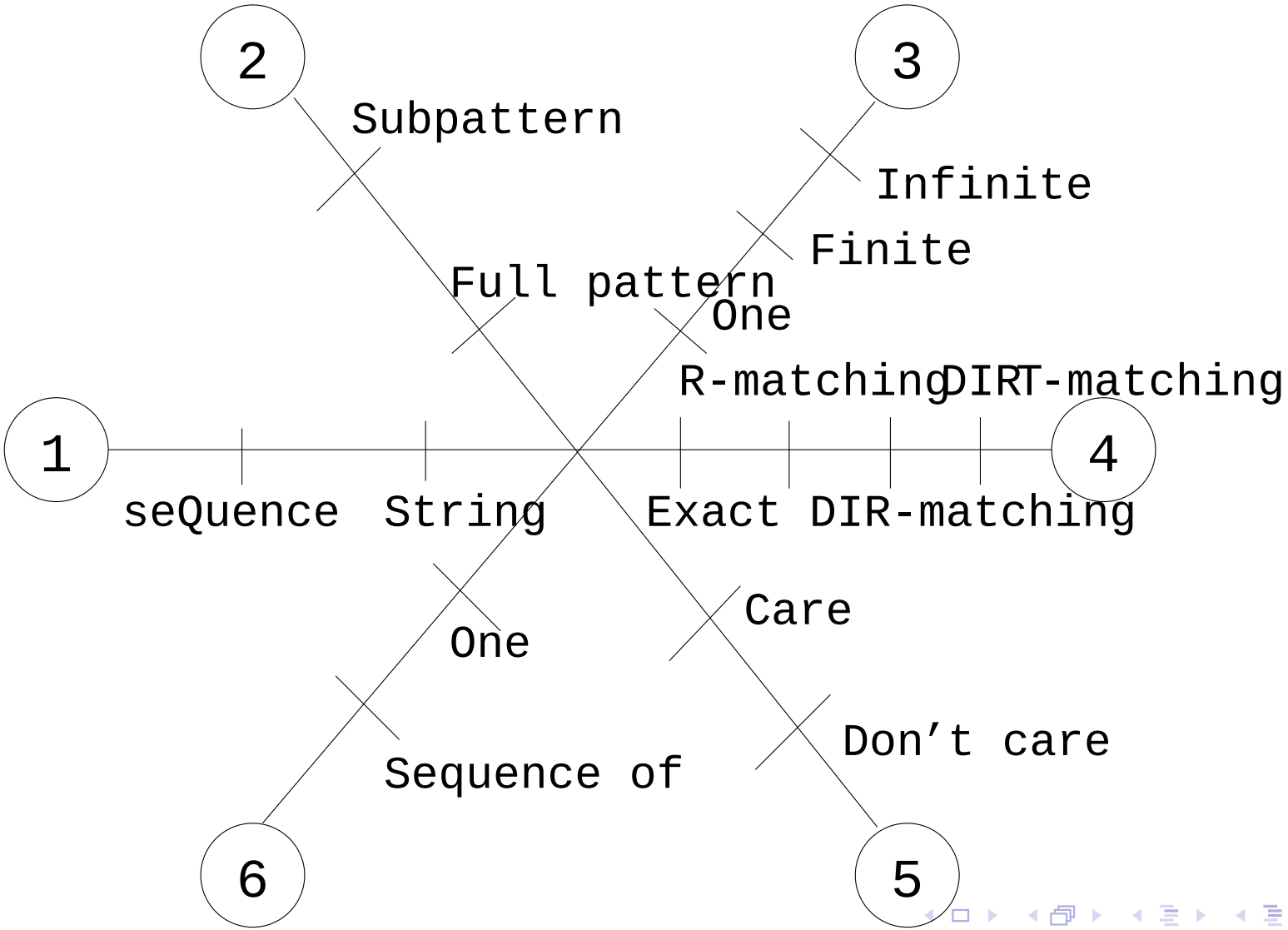
Klasifikace vyhledávacích problémů

Definice: Nechť $T = t_1 t_2 \ldots t_n$ a vzorek $P = p_1 p_2 \ldots p_m$. Možno se například ptát:

1. je P podřetěz T?
2. je P podsekvence z T?
3. je podřetěz či podsekvence P v T?
4. je P v T takový, že $D(P, X) \leq k$ pro $k < m$, kde $X = t_i \ldots t_j$ je částí T (D je R, DIR či $DIRT$)?
5. je řetěz P obsahující ***don't care symbol*** Ø (*) v T?
6. je sekvence vzorků P v T?

Dále varianty pro více vzorků, plus instance problému pro hledání ano/ne, první výskyt, všechny výskyty nepřekrývající se, všechny výskyty i překrývající se.

6D klasifikace vyhledávacích problémů [MEH] ([MAR])



6D klasifikace vyhledávacích problémů (pokr.)

Dimenze	1	2	3	4	5	6
	S	F	O	E	C	O
	Q	S	F	R	D	S
			I	D		
				G		

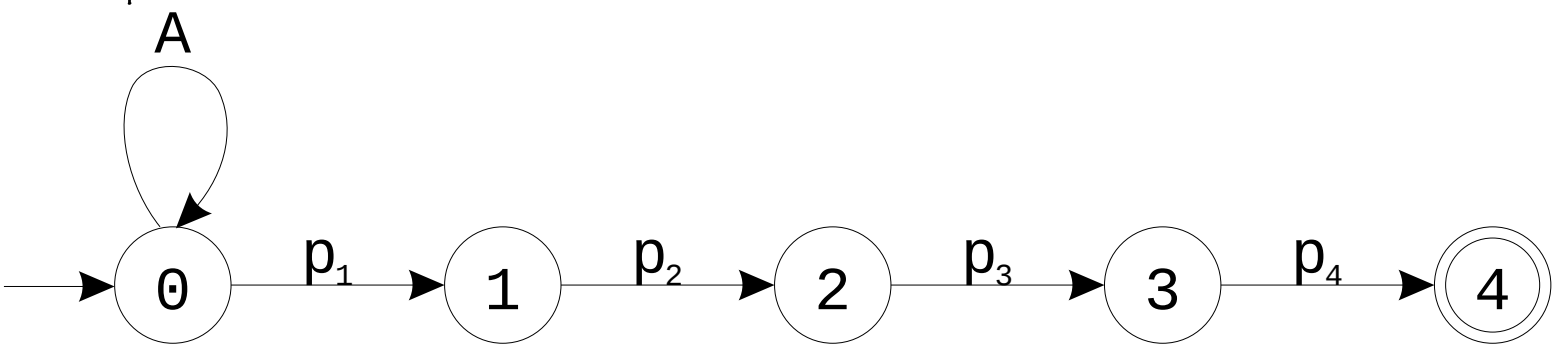
Celkem $2 \times 2 \times 3 \times 4 \times 2 \times 2 = 192$ vyhledávacích problémů rozklasifikovaných v šestirozměrném prostoru.

Například SFO??? značí všechny VS pro hledání jednoho (celého) řetězce.

Pro všechny tyto problémy se naučíme vytvářet vyhledávací NKA.

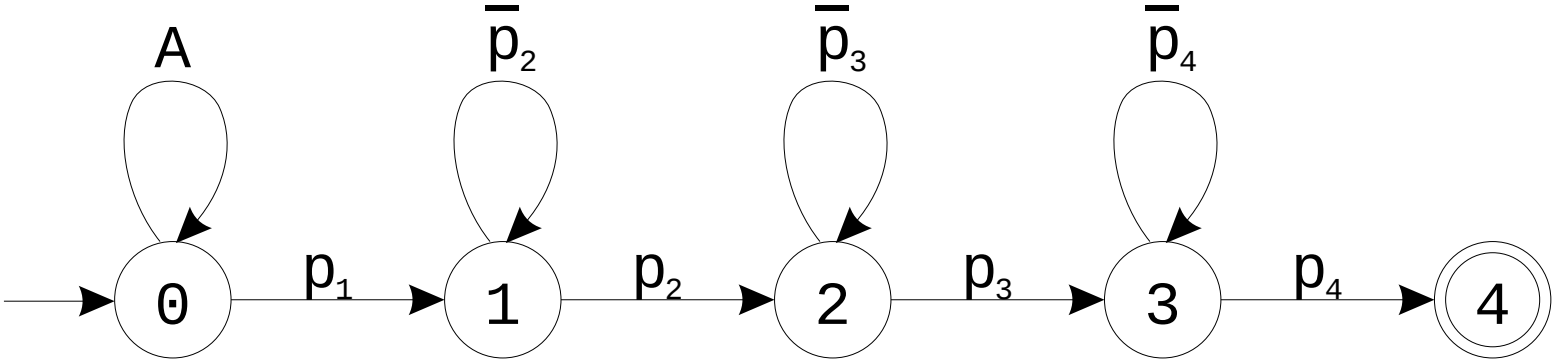
Příklady vytváření VS

Příklad: Nechť $P = p_1 p_2 p_3 \ldots p_m$, $m = 4$, A je jakýkoliv znak ze Σ.
NKA pro SFOECO:



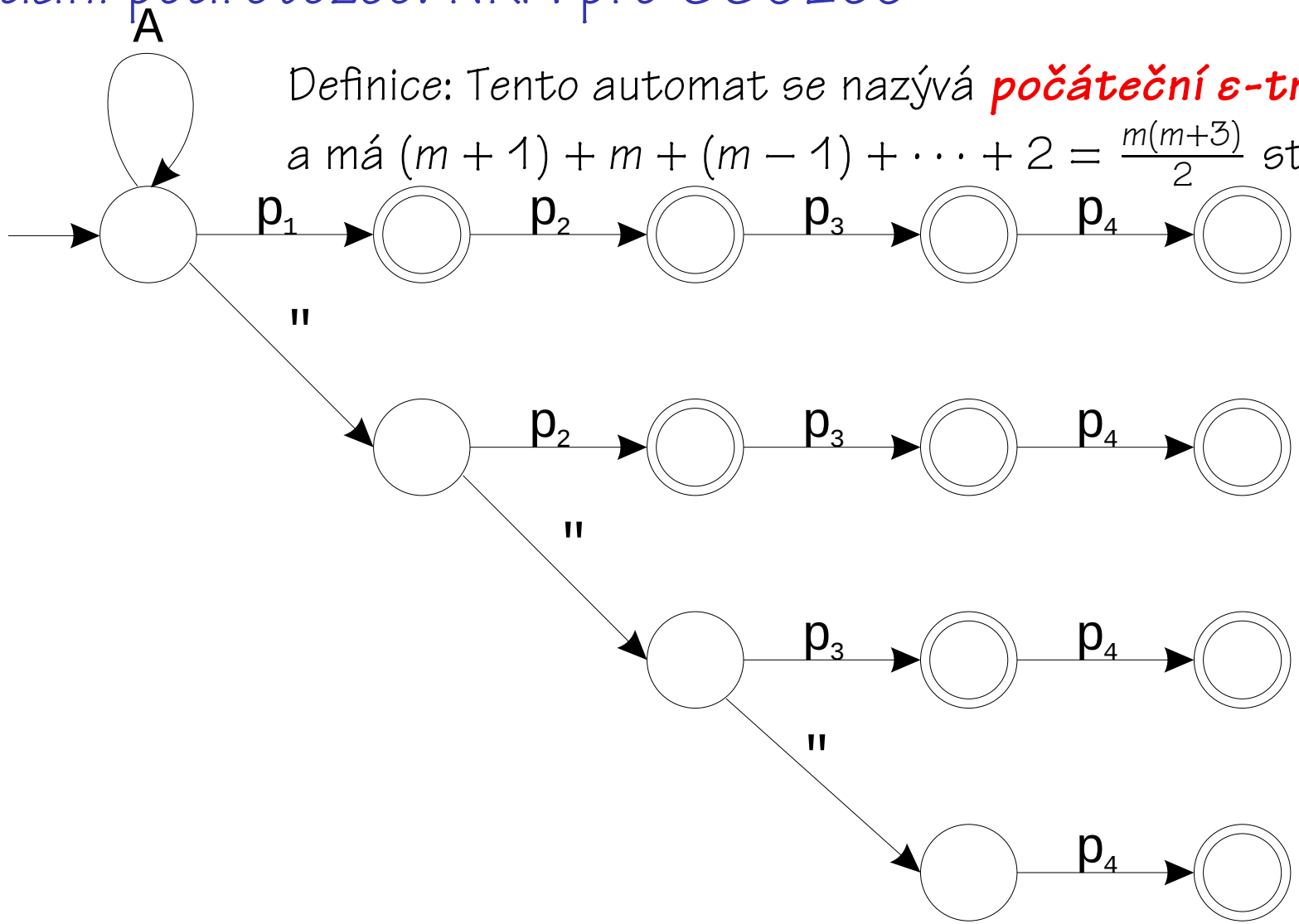
Hledání sekvencí znaků

Příklad: NKA pro QFOECO (seQuence):



$\overline{p}$ je jakýkoliv znak ze Σ kromě p. Automat má $m + 1$ stavů pro vzorek délky m.

Hledání podřetězce: NKA pro SSOECO

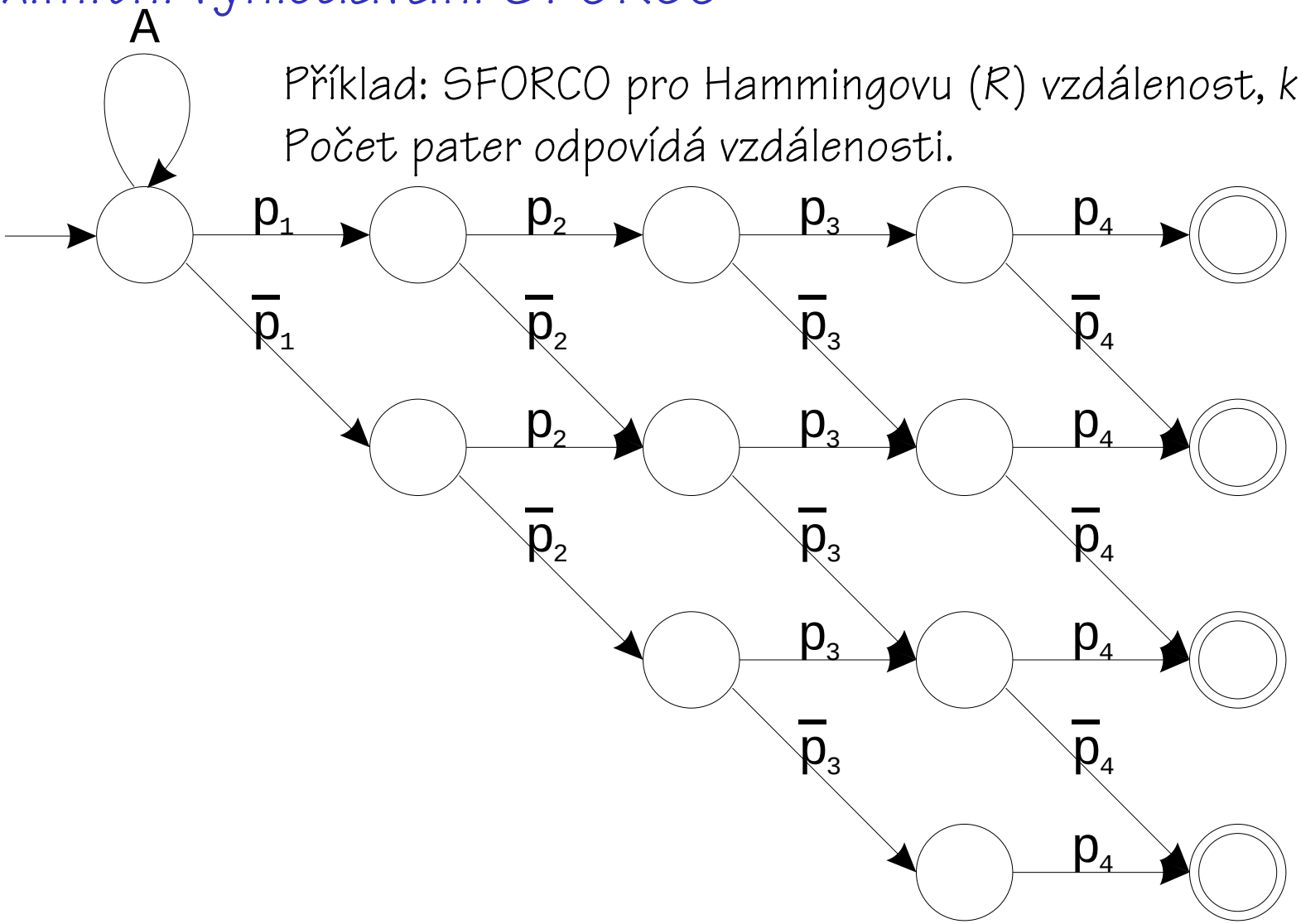


Definice: Tento automat se nazývá **počáteční ε-treelis** a má $(m+1)+m+(m-1)+\cdots+2=\frac{m(m+3)}{2}$ stavů.

Hledání podsekvence

Příklad: NKA pro QSOECO je podobný, jen přidáme cykly pro nesrovnalé znaky a ε přechody ke všem existujícím dopředným přechodům (nebo zřetězíme automat m-krát).
Definice: Automat pro QSOECO se nazývá ***ε-treelis***.

Proximitní vyhledávání SFORCO



Příklad: SFORCO pro Hammingovu (R) vzdálenost, $k \leq 3$:
Počet pater odpovídá vzdálenosti.

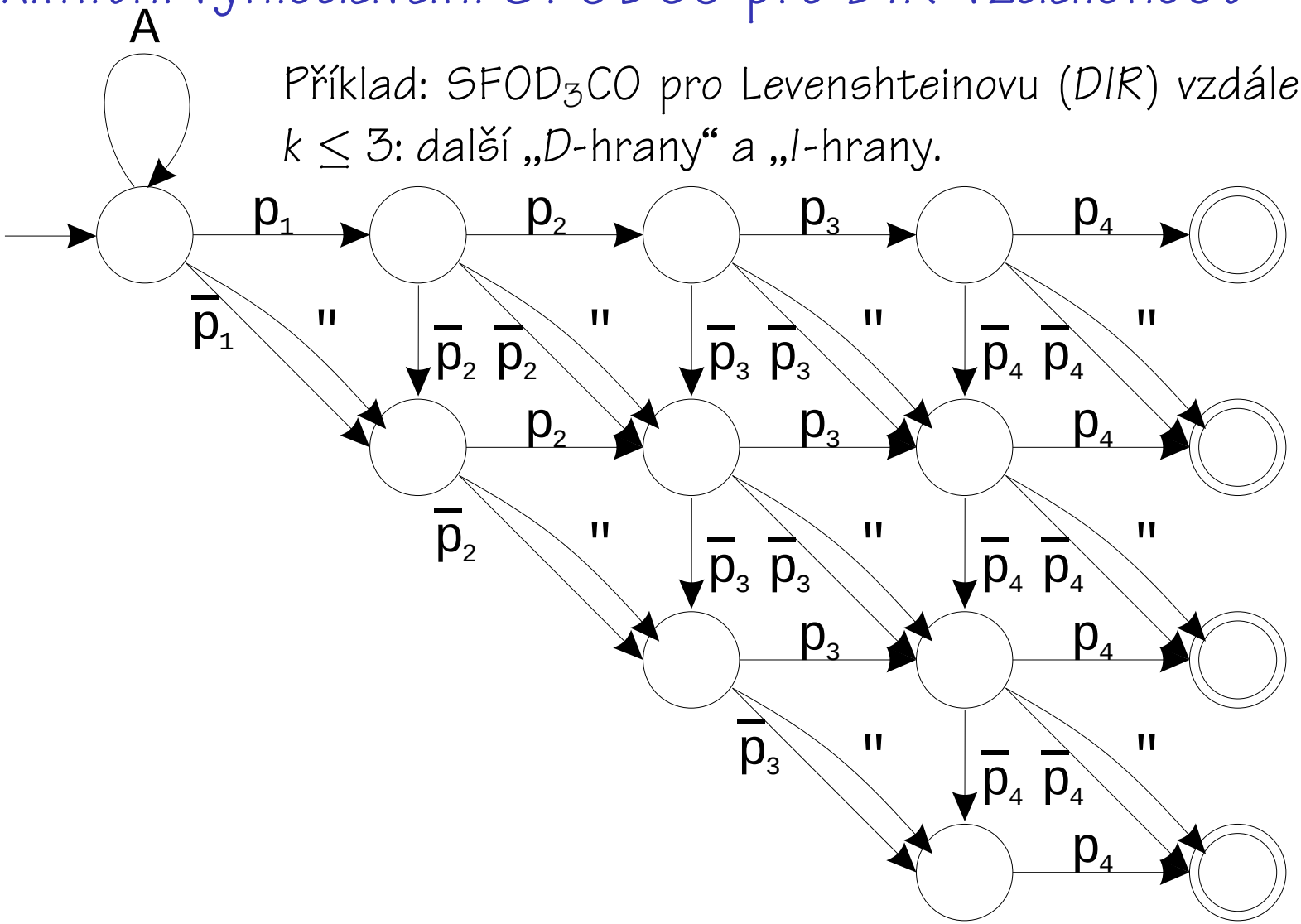
Proximitní vyhledávání SFORCO

Definice: Tento automat se nazývá **R-treelis**, a má $(m+1)+m+(m-1)+\cdots+(m-k+1) = (k+1)(m+1-\frac{k}{2})$ stavů.

Číslo patra koncového stavu odpovídá vzdálenosti nalezeného řetězce od vzoru.

Proximitní vyhledávání SFODCO pro DIR-vzdálenost

Příklad: $SFOD_3CO$ pro Levenshteinovu (*DIR*) vzdálenost, $k \leq 3$: další „*D*-hrany“ a „*I*-hrany.



SFOGCO

Pro DIRT-vzdálenost ještě přidáme nové stavy do automatu SFODCO odpovídající operaci transpozice a odpovídající dvojici hran pro každou transpozici.

Animační program pana Pojera pro diskutované vyhledávací stroje je ke stažení z webové stránky předmětu a je také instalován v B311.

Simulace NKA nebo determinizace? Hybridní přístup.

Pražský stringologický klub a jeho konference.

Část X

Indexové metody

Osnova (Týden šestý)

1. Uzavření kapitoly vyhledávání bez předzpracování textu.
2. Vyhledávání s předzpracováním textu; indexové metody.
3. Metody indexování.
4. Automatické indexování, konstrukce tezauru.
5. Způsoby implementace indexu.

Vyhledávání s předzpracováním textu

Velké množství textů? Předzpracování textu!

Základní pojmy

- index, indexové metody, indexový soubor, indexsekvenční soubor
- hierarchické členění textu, ***značkování*** textu, ***hypertext***
- otázky uložení seznamu slov (***lexikon***) a seznamu výskytů (hitů), jejich aktualizace

Vyhledávání s předzpracováním textu

- granularita položek indexu: dokument – odstavec – věta – slovo

	slovo1	slovo2	slovo3	slovo4
dok1	1	1	0	1
dok2	0	1	1	1
dok3	1	0	1	1

- ***invertovaný soubor***, transpozice

	dok1	dok2	dok3
slovo1	1	0	1
slovo2	1	1	0
slovo3	0	1	1
slovo4	1	1	1

Vyhledávání v indexu

Indexové metody

- Uspořádání slov (primární klíč) v indexu $\longrightarrow$ ***binární vyhledávání***
  Časová složitost vyhledávání jednoho slova v indexu: n délka indexu, V délka vzorku
  $O(V \times \log_2(n))$
- Vyhledávání k slov, vzorek $p = v_1, \ldots, v_k$
  $k \ll n \Rightarrow$ ***opakované binární vyhledávání***
  s průměrná délka vzorku, složitost?
  $O(s \times k \times \log_2 n)$
- Pokud k a i srovnatelné: ***metoda dvojitého slovníku***.
- ***Hašování***.

Rychlost $O(n)$ ani $O(\log n)$ však obvykle nedostačuje, je třeba $O(1)$.

Implementace indexových systémů I

Pro implementaci indexu je klíčová volba vhodných datových struktur a algoritmů.

- Použití invertovaného souboru:

slovo1	1	0	1
slovo2	1	1	0
slovo3	0	1	1
slovo4	1	1	1

- Použití seznamu dokumentů:

slovo1	1, 3
slovo2	1, 2
slovo3	2, 3
slovo4	1, 2, 3

- Souřadnicový systém s ukazateli má 2 části: slovník s ukazateli do seznamu dokumentů a zřetězený seznam ukazatelů na dokumenty

Metody indexování

Vytváření rejstříku – indexování

- ruční vs. automatické, pros/cons
- ***stop-list*** (slova s gramatickým významem – spojky, předložky, ...)
  1. neřízené
  2. řízené (speciální slovník slov: stanovení indexovacího jazyka) – ***pass-list***, tezaurus.
- synonyma a slova příbuzná.
- flektivní jazyky: vytváření rejstříku s jazykovou podporou – ***lemmatizace***.

Analýza textu – výběr slov do indexu

Frekvence výskytu slov je při identifikaci dokumentu významná.

Frekvenční slovník angličtiny:

1	the	69971	0.070
2	of	36411	0.073
3	and	28852	0.086
4	to	26149	0.104
5	a	23237	0.116
6	in	21341	0.128
7	that	10595	0.074
8	is	10099	0.088
9	was	9816	0.088
10	he	9543	0.095

- ***Zipfův zákon*** (princip nejmenšího odporu)
  pořadí × frekvence ≅ konstanta
- ***Kumulativní podíl používaných slov*** $KPS = \frac{\sum_{pořadí=1}^{N} frekvence_{pořadí}}{počet\ slov\ textu}$
- Pravidlo 20–80: 20 % nejfrekventovanějších slov tvoří 80 % textu [MEI

Metoda automatického indexování

Metoda automatického indexování je založená na odvození významnosti slov z jejich frekvencí (cf. Collins-Cobuild dictionary); slova s nízkou a vysokou frekvencí jsou vyloučena:
VSTUP: n dokumentů
VÝSTUP: seznam slov vhodných pro vytvoření indexu

1. Spočteme frekvenci $FREQ_{ik}$ pro každý dokument $i \in \langle 1, n \rangle$ a každé slovo $k \in \langle 1, K \rangle$ [K je počet různých slov ve všech dokumentech].
2. Spočteme $TOTFREQ_k = \sum_{i=1}^{n} FREQ_{ik}$.
3. Vytvoříme frekvenční slovník pro slova $k \in \langle 1, K \rangle$.
4. Stanovíme práh pro vyloučení velmi frekventovaných slov.
5. Stanovíme práh pro vyloučení slov s nízkou frekvencí.
6. Zbývající slova zařadíme do indexu.

Problematika určení prahů [MEL, obr. 4.20].

Vytváření rejstříku – lemmatizace

Využití morfologie při vytváření slovníku

- kmen/ kořen slova (učit, uč);
- program lemma (modul korpus, příklady), ajka (abin);
- technika vzorů pro určení kmene.

Vytváření rejstříku – tezaurus

- Tezaurus – slovník, obsahující hierarchické a asociativní vztahy a vztahy ekvivalence mezi jednotlivými termíny.
- Vazby mezi termíny/lemmaty:
  - **synonyna** – vazba na standardní termín (viz);
  - vazba na příbuzný termín (viz také); RT – related term;
  - vazba na obecnější termín; BT – broader term;
  - vazba na užší termín; NT – narrower term;
  - **hyperonyma** (auto:dopravní prostředek); **hyponyma** (pták:sojka); **meronymum** (dveře:zámek); **holonyma** (ruka:tělo); **antonyma** (dobrý:zlý).
- Pes/Fík, Havel/president

Konstrukce tezauru

ručně/ poloautomatizovaně

- heuristiky konstrukce tezauru:
  - hierarchická struktura/y tezauru
  - *oborové* tezaury, sémantika závislá kontextově (př. pole, strom v informatice)
  - sdružování termínů s podobnou frekvencí
  - vyloučení termínů s vysokou frekvencí
- šíře aplikací tezauru a lemmatizátoru: kromě indexování spelování, základ pro grammar checker, plnotextové vyhledávání.
- projekty WORDNET, EUROWORDNET
- `module add wordnet; wn`
  `wn faculty -over -simsn -coorn`

Hierarchický tezaurus

- Vytváření znalostní báze pro přesné vyhodnocení relevance dokumentů.
- ***topic*** – zpracování sémantických map termínů. Visual Thesaurus `http://www.visualthesaurus.com`
- Tovek Tools, Verity.

Část XI

Exkurs do počítačové lingvistiky

Počítačová lingvistika

Úrovně zpracování (indexování) textů v přirozeném jazyce

- vyhledávání řetězců – slova jsou řetězce písmen.
- slovotvorba – morfologická analýza.
- gramatika (CFG, DFG) – syntaktická analýza.
- význam vět (TIL) – sémantická analýza.
- kontext – pragmatická analýza.
- plné porozumění a schopnost komunikace – informace.

Corpus Query Procesor

základní dotazy

- „Havel“;

45: Český prezident Václav <Havel> se včera na
89: jak řekl Václav <Havel> , každý občan
248: více než rokem <Havel> řekl Pravda vítězí

regulární výrazy

- „Pravda|pravda“;
- „(P|p)ravda“;
- „(P|p)ravd[a,u,o,y]“;
- „pravd.*“; „pravd.+“; „post?el“;

posloupnost slov

- „prezident(a|u)“ „Havl(a|ovi)“;
- „a tak“;
- „prezident“; []* „Havel“;
- „prezident“ („republiky“ „Vaclav“)? „Havel“;

Corpus Query Procesor

dotazy na poziční atributy

- [word = „Havel“];
- [lemma = „prezident“] []* [lemma = „Havel“];
- …ženu prezidenta Havla …
  [lemma = „hnát“] [] [lemma = „Havel“];
- [word = „žen(u|eme)“ & lemma !=„žena“]; | … or
  ! … not

některé další možnosti

- [lemma = „prezident“] []* [lemma = „Havel“] within s; …10, 3 s
- [lemma = „Havel“] within 20 </s>„Pravda“
- <s>a:[word= „Žena|Muž|Člověk“] []* [lemma = a.lemma]

Rub a líc relevantního vyhledávání

Velká výpočetní síla dnešních počítačů umožňuje:

- efektivní uložení velkého objemu textových dat (komprese, indexování);
- efektivní vyhledávání textových řetězců.

To všechno využije člověk sedící za počítačem, aby z takto zpracovaných textů získal informace, které ho zajímají. Opravdu?

Příklad: V textové databázi je uloženo několik posledních ročníků denního tisku. Rád bych získal informace o prezidentu Václavu Havlovi.
a/>HAVEL
b/>přesnější dotazy
c/...
...

Počítač (výpočetní síla)	+	člověk (inteligence)	=	hodnotné informace

Cíl nás všech →

\+ čas
\+ peníze

přenést co největší část inteligence (času, peněz, ...) na počítač.

Osnova (Týden sedmý)

- Exkurs do počítačové lingvistiky.
- Korpusová lingvistika jako příklad TIS.
- Vyhledávací metody s předzpracování textu i vzorku (dotazu).
- Google jako příklad TIS.

Rub a líc relevantního vyhledávání

Úrovně zpracování (indexování, adresování) textů v přirozeném jazyce

informace	ideál ideálů	ne	Vyhledávání	
pragmatická analýza	kontext	ne	informací	Spravný
semantická analýza	význam vět TIL	v začátcích	Kontrola	překlad
syntaktická analýza	gramatika CFG, DCG	částečně	pravopisu	
morfologická analýza	slovotvorba lemma	ano	Kontrola	Překlad jedn.
slova jsou řetězce písmen	vyhledávání řetězců	ano		

Rub a líc získávání informací z přirozeného jazyka

Víme opravdu, co je informace obsažená v textu v přirozeném jazyce?

- František Novák váží 100 kg. ⟶ **RDB**
  objekt vlastnost hodnota — *atribut1, atribut2, …*
- František Novák má rád pivo. ? *klíč hodnota*
  František Novák má rád ↗
  Janu Novotnou
- F. N. je starý poctivec. ⟶ ?
  Jaro propuklo v plné síle.

Slova přirozeného jazyka označují objekty, jejich vlastnosti a vztahy mezi nimi. Na slova a věty je možné pohlížet také jako na „funkce“ svého druhu, definované významem. Textovou informaci (konkrétní objekty o nichž hovoříme, pragmatické informace, …) lze pak chápat jako „výpočet“ těchto funkcí. Tento výpočet je pro nejednoznačnost často nepřesný nebo nemožný.

- Muž, který vystoupil na nejvyšší českou osmitisícovku, je můj vnuk.

Korpusová lingvistika

- **Korpus**: elektronická kolekce textů, často indexovaná lingvistickými značkami.
- Korpus jako textový informační systém: korpusová lingvistika.
- BNC, Penn Treebank, DESAM, PNK, ...; rozsahy řád milionů až miliard pozic (slov), speciální metody nutné.
- Korpusové manažery CQP, GCQP, Manatee/Bonito, `http://www.fi.muni.cz/~pary/korp/`.
- Vnitřní architektura a implementace korpusu.

viz podklady [MAR].

Co je korpus?

Definice: **Korpus** je rozsáhlý, vnitřně strukturovaný ucelený soubor textů v přirozeném jazyce elektronicky uložený a zpracovatelný.

Historie/motivace

- Indiánské jazyky nemají písmo – pro nalezení gramatiky potřeba nejprve sepsat mluvené slovo.
- 1967 – 1. korpus v U. S. A. (Kučera, Francis) 1 000 000 slov.
- Noam Chomsky – odmítá korpusy.
- Dnes – masové rozšíření.

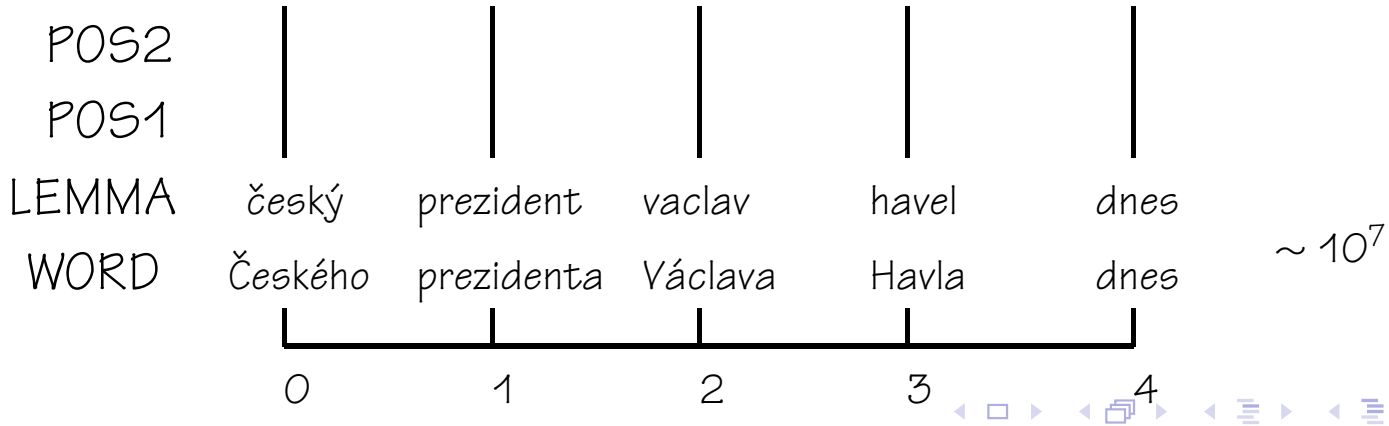
Korpusy na FI

- WWW stránka Pavla Rychlého (~PARY) odkaz na základní informace. Bonito, Manatee.
- IMS CORPUS WORKBENCH – sada nástrojů pro efektivní kódování, reprezentaci a dotazování nad velkými textovými soubory.

Logický pohled na korpus

Posloupnost slov na očíslovaných pozicích (první slovo, *n*-té slovo), kterým jsou přidány ***značky*** (přidávání značek nazývané ***značkování*** korpusu). Značky nesou morfologické, gramatické a libovolné další údaje o daném slovu. To vede k obecnějšímu pojmu ***pozičních atributů***, které jsou nejdůležitější značkovací typ. Atributy z této třídy mají hodnotu (řetězec) na každé korpusové pozici. Na každou z nich je navázáno jedno slovo textu jako základní a poziční atribut word. Kromě tohoto atributu mohou být s každou pozicí svázány další poziční atributy libovolného textu představující morfologické a jiné značky.

Strukturální atributy – věty, odstavce, nadpis, článek, SGML.



Vnitřní architektura korpusu

Dva klíčové pojmy interní reprezentace pozičních atributů jsou:

- **Jednotná reprezentace:** položky jsou pro všechny atributy zakódovány jako čísla typu integer, kde stejné hodnoty mají stejný číselný kód. Posloupnost položek je pak reprezentována jako posloupnost integerů. Interní reprezentací atributu word (stejně jako každého jiného poz. atributu) je `array(0..p-1) of Integer`, kde `p` je počet pozic v korpusu.
- **Invertovaný soubor:** pro posloupnost čísel reprezentující posloupnost hodnot daného atributu je vytvořen invertovaný soubor. Tento soubor pro každou hodnotu (lépe kód hodnoty) obsahuje množinu výskytů položky v pozičním atributu. Inverzní soubor je potřebný pro vyhledávání, protože přímo ukazuje množinu výskytů dané položky, výskyty pak mohou být počítány v jednom kroku.

Vnitřní architektura korpusu (pokr.)

Soubor se zakódovanými hodnotami atributu i inverzní soubor mají pomocné indexové soubory. Pro soubor se zakódovanými hodnotami:

- První datovou strukturou je **seznam položek** či „lexikon“: ten obsahuje množinu rozdílných hodnot. Interně se jedná o množinu řetězců vyskytujících se v posloupnosti položek, kde znak `Null` (osmičkové 000) je vložen za každé slovo. Seznam položek už definuje kód pro každou položku, protože předpokládáme, že první položka v seznamu má kód 0, následující 1 atd.
- Pro vyhledávání řetězce v našem seznamu je užitečné mít **index na pozici začátku řetězce** v tomto souboru. Tento index pro každý kód položky dává konverzi z kódu položky na relativní adresu (v bytech) v seznamu položek…

Vnitřní architektura korpusu (pokr.)

Pro invertovaný soubor existují tři datové struktury:

- První je samostatný invertovaný soubor, který obsahuje množinu korpusových pozic.
- Druhý je index do tohoto souboru. Tento index vrací pro každý kód položky vstupní bod náležící výskytu v inverzním souboru.
- Třetí je tabulka frekvence kódu položek, která dává pro každý kód položky číslo výskytu kódu v korpusu (které je samozřejmě stejné jako velikost množiny výskytů).

Implementace indexových systémů

Přehled možných přístupů a implementací

- Invertovaný soubor – indexový soubor s bitovým vektorem.
- Použití seznamu dokumentů ke každému klíčovému slovu.
- Souřadnicový systém s ukazateli [MEL, obr. 4.18, strana 46].
- Indexace korpusových textů: Finlib
  http://www.fi.muni.cz/~pary/dis.pdf
  viz podklady [MAR].
- Použití Eliasových kódů pro komprimaci seznamu hitů.

Implementace indexových systémů (pokr.)

- Efektivní uložení indexu/slovníku [lemmat]: ***packed trie***, Patricia tree, a další stromové struktury.
- Syntactic neural network (S. M. Lucas: Rapid best-first retrieval from massive dictionaries, Pattern Recognition Letters 17, p. 1507–1512, 1996).
- Komerční implementace: Verity engine, webové vyhledávače většinou svůj klíč k úspěchu až na výjimky tají.

Reprezentace slovníku KA I

Článek M. Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing viz podklady [MAR]

- Reprezentace slovníku konečným automatem.
- Nejednoznačnosti, sjednocení minimalizovaných deterministických automatů.
- Příklad: done,do.V3:PP
  done,done.A0
- Morfologický slovník jako seznam dvojic [slovní tvar, lemma].
- Kompaktace uložení datové struktury automatu (Liang, 1983).
- Kompresní poměr až 1:20 při lineárním přístupu (vzhledem k délce slova).

Reprezentace slovníku KA II

- Převodník (transducer) pro reprezentaci slovníku.
- Deterministický převodník s 1 výstupem (subsequential transducer) pro reprezentaci slovníku včetně jednoho řetězce na výstupu (informace o morfologii, dělení slov,...).
- Deterministický převodník s p výstupy (p–subsequential transducer) pro reprezentaci slovníku včetně více řetězců na výstupu (víceznačnosti).
- Determinizace převodníku obecně neuskutečnitelná (třída deterministických převodníků s výstupem je vlastní podtřída nedeterministických převodníků); pro účely zpracování přirozeného jazyka však obvykle nenastává (nejsou zde cykly).

Reprezentace slovníku KA III

- Přidání stavu do převodníku odpovídajícího (w_1,w_2) bez porušení determiničnosti: nejprve stav pro (w_1,ε), pak s výsledným stavem konečný stav s výstupem w_2.
- Efektivní metoda, rychlá, leč není minimální; existují minimalizační algoritmy, které vedou na prostorově úsporná řešení.
- Postup: rozdělení slovníku na části, vytvoření det. převodníků s p výstupy, jejich miminalizace, pak deterministické sjednocení převodníků a minimalizace výsledného.
- Další použití i při efektivní indexaci, rozpoznávání řeči, atd.

Vyhledávací metody IV.

Předzpracování textu i vzorku (dotazu): drtivá většina dnešních TIS.
Typy předzpracování:

- statistiky *n*-gramů (fragmentové indexy).
- speciální algoritmy pro zpracování indexů (kódování, komprese) a vyhodnocení relevance (PageRank u Google).
- využití metod zpracování přírozeného jazyka (morfologie, syntaktická analýza, sémantické databáze) a agregace informací z více zdrojů (systémy AnswerBus, START).
- signaturové metody.

Relevance

Definice: **Relevance** (odpovědi na dotaz) je míra rozsahu, kterým se vybraný dokument shoduje s požadavky na něj kladenými.
Ideální odpověď ≡ skutečná odpověď
Definice: **Koeficient úplnosti R (recall)** $R = \frac{m}{n}$, kde m je počet vybraných relevantních záznamů a n je počet všech relevantních záznamů v TIS.
Definice: **Koeficient přesnosti P (precision)** $P = \frac{m}{o}$, kde o je počet všech vybraných záznamů dotazem.
Chceme docílit maximum R i P, tradeoff.

Standardní hodnoty: 80 % pro P, 20 % pro R.

Kombinace úplnosti a přesnosti: **koeficient** $F_b = \frac{(b^2+1)PR}{b^2P+R}$. ($F_0 = P$, $F_\infty = R$, při $F_1 = F$ P a R váženy stejně).

Fragmentový index

Vyhledávání pomocí fragmentových indexů

- Fragment ybd je v angličtině pouze ve slově molybden.
- Výhody: pevný slovník fragmentů, odpadají problémy s aktualizací.
- Nevýhody: závislost na jazyce a tématické oblasti, snížená přesnost vyhledávání.

Gooooooooooooogle

Příklad anatomie globálního (hyper)textového informačního systému (www.google.com).

- Jeden z mála kvalitních vyhledávacích strojů, jehož základní principy a architektura jsou aspoň v principech známy – proto detailnější rozbor dle článku [GOO]
  http://www7.conf.au/programme/fullpapers/1921com1921.ht
- Několik inovativních konceptů: PageRank, ukládání lokálního komprimovaného archívu, výpočet relevance z textů hypetextových odkazů, indexace PDF, Google File System, Google Link...
- Anatomie systému. viz podklady [MAR]

Google: Relevance

Klíčové je určení relevance.

- Využití značek textu a typografie webu pro výpočet relevance termů dokumentu.
- Využití textu hypertextových odkazů na dokument odkazujících.

Google: PageRank

- **PageRank**: objektivní míra důležitosti stránky na základě citační analýzy (vhodné pro utřídění odpovědí na dotaz, tedy určení relevance stránek).
- Nechť na stránku A ukazují stránky, $T_1, \ldots, T_n$ (citace), a nechť $0 < d < 1$. Nechť $C(A)$ je počet odkazů na stránce A. PageRank

$$PR = (1 - d) + d \left(\frac{PR(T_1)}{C(T_1)} + \ldots \frac{PR(T_n)}{C(T_n)} \right)$$

- PageRank se dá spočítat jednoduchým iterativním algoritmem (pro desítky milionů stránek za hodiny na běžném PC).
- PageRank je pravděpodobnostní rozdělení nad webovými stránkami.
- Motivace s náhodným surfařem, faktor d.

Datové struktury Google

- Uložení signatur souborů
- Uložení lexikonu
- Uložení seznamu hitů
- Google File System

Část XII

Kódování

Osnova (Týden osmý)

- Dokončení anatomie Google.
- Kódování.
- Entropie, redundance.
- Universální kódování celých čísel.

Kódování – základní pojmy

Definice: **Abeceda A** je konečná neprázdná množina symbolů.
Definice: **Slovo** (**řetězec**, **zpráva**) nad A je posloupnost symbolů z A.
Definice: **Prázdný řetězec ε** je prázdná posloupnost symbolů.
Množinu všech neprázdných slov nad A značíme A^+.
Definice: **Kód** K je trojice (S, C, f), kde S je konečná množina **zdrojových jednotek**, C je konečná množina **kódových jednotek**, $f : S \longrightarrow C^+$ je injektivní zobrazení.
f lze rozšířit na $S^+ \longrightarrow C^+$: $F(S_1 S_2 \ldots S_k) = f(S_1) f(S_2) \ldots f(S_k)$.
C^+ se někdy nazývá **kód**.

Základní vlastnosti kódů

Definice: $x \in C^+$ je ***jednoznačně dekódovatelný*** vzhledem k f, jestliže existuje maximálně jedna posloupnost $y \in S^+$ taková, že $f(y) = x$.
Definice: Kód $K = (S, C, f)$ je ***jednoznačně dekódovatelný*** jestliže jsou jednoznačně dekódovatelné všechny řetězy v C^+.
Definice: Kód se nazývá ***prefixový***, jestliže žádné kódové slovo není prefixem jiného.
Definice: Kód se nazývá ***sufixový***, jestliže žádné kódové slovo není sufixem jiného.
Definice: Kód se nazývá ***afixový***, jestliže je prefixový i sufixový.
Definice: Kód je ***úplný*** jestliže po přidání libovolného dalšího kódového slova vznikne kód, který není jednoznačně dekódovatelný.

Základní vlastnosti kódů

Definice: ***Blokový kód délky n*** je takový kód, při kterém všechna kódová slova mají délku n.

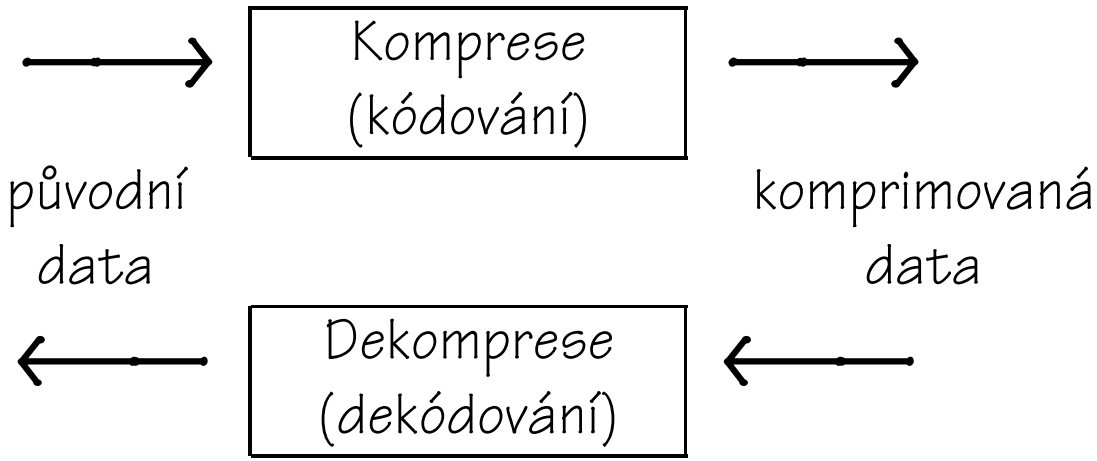
Příklad: blokový ? prefixový

blokový $\Rightarrow$ prefixový, ale ne naopak.

Definice: Kód $K = (S, C, f)$ nazveme ***binární***, jestliže $|C| = 2$.

Komprese a dekomprese

Definice: ***Komprese*** (***kódování***), ***dekomprese*** (***dekódování***):



Definice: ***Kompresní poměr*** je poměr délky komprimovaných dat a délky původních dat.

Příklad: Navrhněte binární prefixový kód pro desítkové číslice, jestliže se často vyskytují čísla 3 a 4, a zřídka 5 a 6.

Entropie a redundance I

Nechť Y je náhodná proměnná s pravděpodobnostním rozdělením $p(y) = P(Y = y)$. Pak matematické očekávání (střední hodnota) $E(Y) = \sum_{y \in Y} yp(y)$.

Nechť $S = \{x_1, x_2, \ldots, x_n\}$ množina zdrojových jednotek a nechť pravděpodobnost výskytu jednotky x_i v informačním zdroji $\mathbf{S}$ je p_i pro $i = 1, \ldots, n$, $n \in N$.

Definice: ***Entropie informačního obsahu jednotky x_i*** (míra množství informace resp. neurčitosti) je $H(x_i) = H_i = -\log_2 p_i$ bitů.

Zdrojová jednotka s větší pravděpodobností nese méně informace.

Entropie a redundance II

Definice: ***Entropie informačního zdroje S*** je $H(\mathbf{S}) = -\sum_{i=1}^{n} p_i \log_2 p_i$ bitů. Platí, že $H(\mathbf{S}) = \sum_{y \in Y} p(y) \log \frac{1}{p(y)} = E\left(\log \frac{1}{p(Y)}\right)$.

Definice: ***Entropie zdrojové zprávy*** $X = x_{i_1} x_{i_2} \ldots x_{i_k} \in S^+$ ***informačního zdroje S*** je $H(X, \mathbf{S}) = H(X) = \sum_{j=1}^{k} H_i = -\sum_{j=1}^{k} \log_2 p_{i_j}$ bitů.

Definice: ***Délka l(X) zakódované zprávy X***

$l(X) = \sum_{j=1}^{k} |f(x_{i_j})| = \sum_{j=1}^{k} d_{i_j}$ bitů.

Věta: $l(X) \geq H(X, \mathbf{S})$.

Entropie a redundance III

Axiomatické zavedení entropie viz podklady [MAR], detaily odvození viz
ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kod

Definice: $R(X) = I(X) - H(X) = \sum_{j=1}^{k} (d_{i_j} + \log_2 p_{i_j})$ je ***redundance kódu K pro zprávu X***.

Definice: ***Průměrná délka kódového slova kódu K*** je $AL(K) = \sum_{i=1}^{n} p_i d_i$ bitů.

Definice: ***Průměrná entropie zdroje S*** je

$AE(\mathbf{S}) = \sum_{i=1}^{n} p_i H_i = -\sum_{i=1}^{n} p_i \log_2 p_i$ bitů.

Definice: ***Průměrná redundance kódu K*** je

$AR(K) = AL(K) - AE(\mathbf{S}) = \sum_{i=1}^{n} p_i (d_i + \log_2 p_i)$ bitů.

Entropie a redundance IV

Definice: Kód je ***optimální***, když má minimální redundanci.
Definice: Kód je ***asymptoticky optimální***, pokud pro dané rozložení pravděpodobností se poměr $AL(K)/AE(S)$ blíží k 1, když se entropie blíží ∞.
Definice: Kód K je ***universální***, jestliže existují $c_1, c_2 \in R$ tak, že průměrná délka kódového slova $AL(K) \leq c_1 \times AE + c_2$.
Věta: Universální kód je ***asymptoticky optimální***, jestliže $c_1 = 1$.

Universální kódování celých čísel

Fibonacciho posloupnost a reprezentace

Definice: ***Fibonacciho posloupnost řádu m***

$F_n = F_{n-m} + F_{n-m+1} + \ldots + F_{n-1}$ pro $n \geq 1$.

Příklad: F řádu 2: $F_{-1} = 0$,, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$,...

Příklad: F řádu 3: $F_{-2} = 0$, $F_{-1} = 0$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, $F_4 = 7$, $F_5 = 13$,...

Příklad: F řádu 4: $F_{-3} = 0$, $F_{-2} = 0$, $F_{-1} = 0$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, $F_4 = 8$, $F_5 = 15$,...

Definice: ***Fibonacciho reprezentace*** $R(N) = \sum_{i=1}^{k} d_i F_i$, kde $d_i \in \{0, 1\}$, $d_k = 1$

Věta: Fibonacciho reprezentace není jednoznačná, existuje však taková, že v posloupnosti d_i je nejvýše $m - 1$ po sobě jdoucích jedniček.

Fibonacciho kódy

Definice: **Fibonacciho kód řádu m** $FK_m(N) = d_1 d_2 \dots d_k \underbrace{1 \dots 1}_{m-1 \text{ krát}}$, kde

d_i jsou koeficienty z předchozí věty (jedničky ukončují slovo).

Příklad: $R(32) = 0 * 1 + 0 * 2 + 1 * 3 + 0 * 5 + 1 * 8 + 0 * 13 + 1 * 21$, tedy $F(32) = 00101011$.

Věta: $FK(2)$ je prefixový, universální kód s $c_1 = 2, c_2 = 3$, tedy není asymptoticky optimální.

Universální kódování celých čísel II

Eliasovy kódy

- asymptoticky optimální universální kód, $c_1 = 1$, do $N = 514228$ jsou lepší Fibonacciho kódy řádu n ($FK(n)$).
- unární kód $a(N) = \underbrace{0\,0\ldots0}_{N-1}\,1$.
- binární kód $\beta(1) = 1, \beta(2N + j) = \beta(N)j$, $j = 0, 1$.
- β není jednoznačně dekódovatelný (není prefixový).
- ternární $\tau(N) = \beta(N)\#$.
- $\beta'(1) = \epsilon, \beta'(2N) = \beta'(N)0, \beta'(2N + 1) = \beta'(N)1, \tau'(N) = \beta'(N)\#$.
- γ: každý bit $\beta'(N)$ je vložen mezi dvojici bitů z $a(|\beta(N)|)$.
- příklad: $\gamma(6) = 0\overline{1}0\overline{0}1$
- $C_\gamma = \{\gamma(N) : N > 0\} = (0\{0, 1\})^*1$ je regulární a tedy dekódovatelná konečným automatem.

Universální kódování celých čísel III

- $\gamma'(N) = a(|\beta(N)|)\beta'(N)$ stejné délky (permutace bitů $\gamma(N)$), ale čitelnější
- $C_{\gamma'} = \{\gamma'(N) : N > 0\} = \{0^k 1\{0,1\}^k : k \geq 0\}$ není regulární a dekodér potřebuje čítač
- $\delta(N) = \gamma(|\beta(N)|)\beta'(N)$
- příklad: $\delta(4) = \gamma(3)00 = 01100$
- dekodér δ: $\delta(?) = 0011?$
- ω:

```
K := 0;
while ⌊log2(N)⌋ > 0 do
  begin K := β(N)K;
        N := ⌊log2(N)⌋
  end.
```

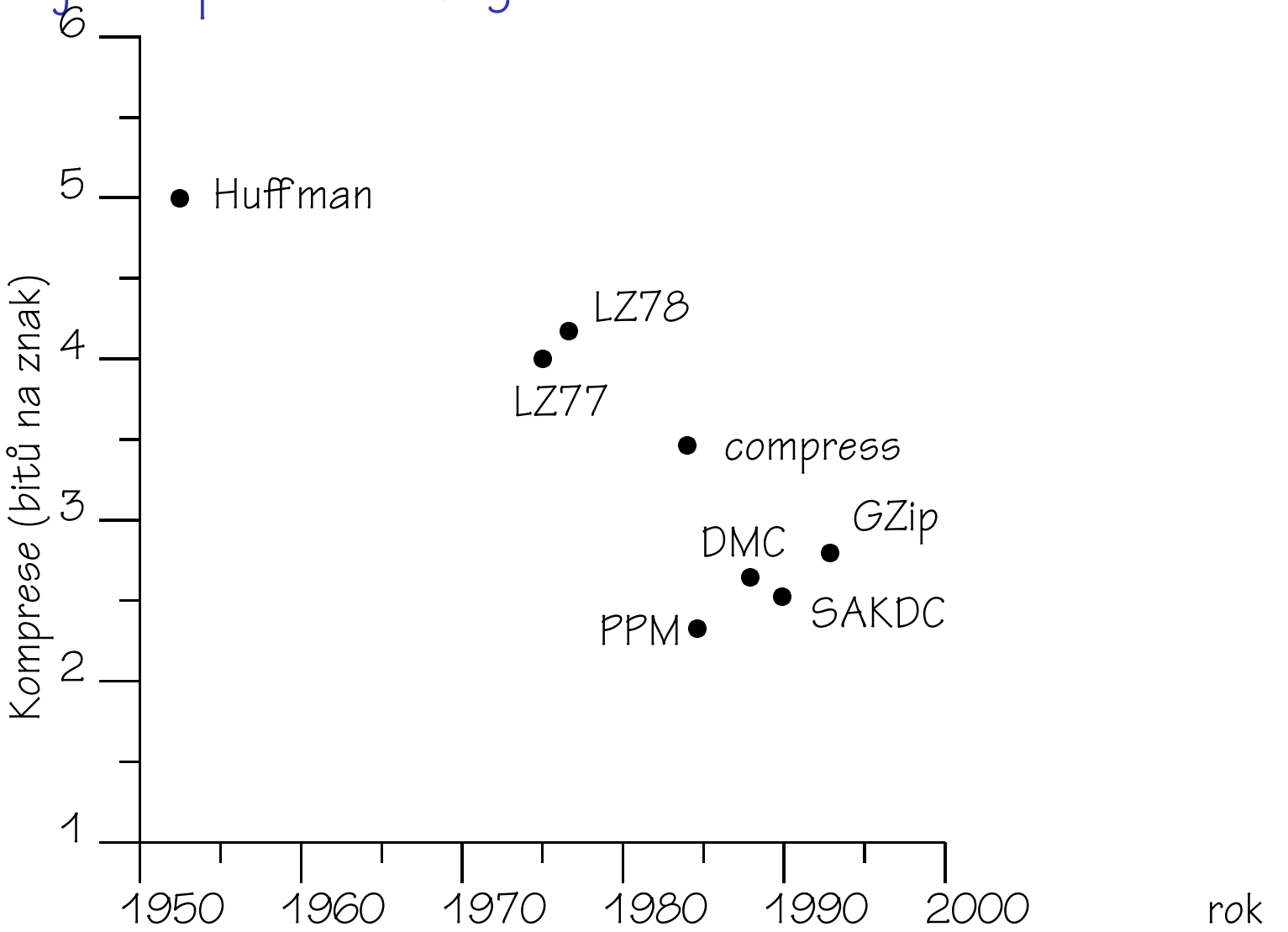
Komprese dat – úvod

- Kódování informace pro komunikační účely; komprese dat.
- Přes bouřlivý vývoj kapacit pro uložení dat stále nedostatek místa.
- Redundance ⟶ konstrukce minimálně redundantního kódu.
- Model dat:
  - struktura – sada jednotek ke komprimaci + kontext výskytů;
  - parametry – pravděpodobnost výskytu jednotlivých jednotek.
- Komprimace:
  1. vytvoření modelu dat;
  2. vlastní kódování.

Komprese dat – vývoj

- 1838 Morse, kód *e* dle četnosti.
- 1949 Shannon, Fano, Weaver.
- 1952 Huffman; 5 bitů na znak.
- 1979 Ziv-Lempel; **`compress`** (Roden, Welsh, Bell, Knuth, Miller, Wegman, Fiala, Green, ...); 4 bity na znak.
- osmdesátá a devadesátá léta PPM, DMC, **`gzip`** (zlib), SAKDC; 2–3 bity/znak
- přelom tisíciletí **`bzip2`**; 2 bity na znak.
- ...?

Vývoj kompresních algoritmů



Predikce a modelování

- redundance (nestejnoměrná pravděpodobnost výskytu zdrojových jednotek)
- kodér, dekodér, model
- statické modelování (model nezávisí na konkrétních datech)
- semiadaptivní modelování (model závisí na datech, 2 průchody, nutnost přenosu modelu)
- adaptivní modelování (1. průchod, model vytvářen dynamicky u kodéra i dekodéra)

Predikce a modelování

- modely řádu 0 – pravděpodobnosti izolovaných zdrojových jednotek (př. Morse, písmeno e)
- modely s konečným kontextem – Markovovy modely, modely řádu n (př. Bach), $P(a|x_1 x_2 \ldots x_n)$
- modely založené na konečných automatech
  - synchronizační řetěz, nesynchronizační řetěz
  - automat s konečným kontextem
  - vhodné pro regulární jazyky, nevhodné pro bezkontextové jazyky, $P(a|q_i)$

Statistické metody komprese I

Znakové techniky

- null supression – nahrazení opakování ≥ 2 znaku null, 255, speciální znak S_c
- run-length encoding (RLE) – $S_c X C_c$ zobecnění na libovolný opakující se znak $\$******55 \longrightarrow \$S_c * 655$
- MNP Class 5 RLE – CXXX *DDDDDBBAAAA* $\longrightarrow$ *5DDDBB4AAA*
- half-byte packing, (EBCDIC, ASCII) SI, SO
- diatomic encoding; nahrazování dvojic znaků jedním
- Byte Pair Encoding, BPE (Gage, 1994)
- pattern substitution
- Gilbert Held: Data & Image Compression

Statistické metody komprese II

- Shannon-Fano, 1949, model řádu 0, prefixový kód, kódový strom.
- kódová slova délky $\lfloor -\log_2 p_i \rfloor$ nebo $\lfloor -\log_2 p_i + 1 \rfloor$
- $AE \leq AL \leq AE + 1$.
- kódový strom (2,2,2,2,4,4,8).
- obecně není optimální, dva průchody kodéru textem, statický $\longrightarrow$

Shannon-Fano kódování

Vstup: posloupnost n zdrojových jednotek $S[i]$, $1 \leq i \leq n$, v pořadí neklesajících pstí.
Výstup: n binárních kódových slov.

```
begin přiřaď všem kódovým jednotkám prázdný řetěz;
      SF-SPLIT(S)
end

procedure SF-SPLIT(S);
begin if |S| ≥ 2 then
      begin rozděl S do posloupností S1 a S2 tak, že obě
                posloupnosti mají přibližně stejnou celkovou pst;
            přidej ke všem kódovým slovům z S1 0;
            přidej ke všem kódovým slovům z S2 1;
            SF-SPLIT(S1); SF-SPLIT(S2);
      end
end
```

Osnova (Týden desátý)

- Huffmanovo kódování.
- Adaptivní Huffmanovo kódování.
- Aritmetické kódování.
- Slovníkové metody.
- Slovníkové metody s restrukt. slovníku.

Huffmanovo kódování

- Huffmanovo kódování, 1952.
- varianty statická a dynamická.
- $AEPL = \sum_{i=1}^{n} d[i]p[i]$.
- optimální kód (ne jediný možný).
- $O(n)$ za předpokladu utříděnosti zdrojových jednotek.
- stabilní rozložení $\longrightarrow$ příprava předem.

Příklad: (2,2,2,2,4,4,8)

Huffmanovo kódování – sourozenecká vlastnost

Definice: Binární strom má ***sourozeneckou vlastnost*** právě tehdy, když

1. každý uzel kromě kořene má sourozence,
2. uzly mohou být seřazeny v pořadí neklesající posloupnosti tak, že každý uzel (kromě kořene) sousedící v seznamu s nějakým uzlem je jeho sourozenec (levý synové budou na lichých místech v seznamu a pravý synové na sudých).

Huffmanovo kódování – vlastnosti Huffmanových stromů

Věta: Binární prefixový kód je Huffmanův $\Leftrightarrow$ má sourozeneckou vlastnost.

- $2n - 1$ uzlů, max. $2n - 1$ možností,
- optimální binární prefixový kód, který není Huffmanův
- $R(X) \leq p_n + 0{,}086$, p_n maximální pravděpodobnost zdrojové jednotky.
- Huffman je úplný, (špatná detekce chyb).
- **afixový kód**, KWIC, levý a pravý kontext, hledání $*X*$

Adaptivní Huffmanovo kódování

- FGK (Faller, Gallager, Knuth)
- potlačení minulosti zapomínacím koeficientem, zaokrouhlování, 1, r, r^2, r^n.
- lineární čas kódování i dekódování vzhledem k délce slova.
- $AL_{HD} \leq 2AL_{HS}$.
- Vitter $AL_{HD} \leq AL_{HS} + 1$.
- implementační detaily, stromová reprezentace kódové tabulky.

Princip aritmetického kódování

- zobecnění Huffmanova kódování (pravděpodobnosti zdrojových jednotek nemusí být záporné mocniny dvou).
- uspořádání na zdrojových jednotkách; **Kumulativní pravděpodobnost** $cp_i = \sum_{j=1}^{i-1} p_j$ zdrojové jednotky x_i s pravděpodobností p_i.
- Výhody:
  - libovolná blízkost entropii.
  - adaptivnost je možná.
  - rychlost.

Slovníkové metody komprese dat

Definice: ***Slovník*** *je dvojice* $D = (M, C)$, *kde* M *je konečná množina slov zdrojového jazyka,* C *zobrazení* M *na množinu kódových slov.*
Definice: $L(m)$ *značí délku kódového slova* $C(m)$ *v bitech, pro* $m \in M$.
Výběr zdrojových jednotek:

- *statický (dohoda na slovníku předem)*
- *semiadaptivní (nutné dva průchody textem)*
- *adaptivní*

Statické slovníkové metody

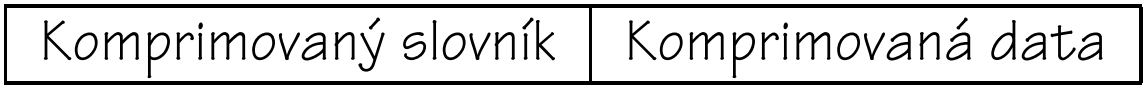
Zdrojová jednotka délky n – n-gramy
Nejčastější bigramy ($n = 2$)

- n pevné
- n proměnné (dle frekvencí výskytu)
- adaptivní

(50 % anglického textu je tvořeno asi 150 nejfrekventovanějšími slovy)
Nevýhody:

- nejsou schopny reagovat na rozdělení pravděpodobností komprimovaných dat
- předem připravený slovník

Semiadaptivní slovníkové metody



Výhody: rozsáhlá data (slovník je malá část dat – korpusy; CQP).

Semiadaptivní slovníkové metody – postup vytvoření slovníku

1. Určí se frekvence N-gramů pro $N = 1, 2, \ldots$.
2. Slovník se inicializuje vložením unigramů.
3. Do slovníku se postupně přidávají N-gramy ($N > 1$) s největší frekvencí. Při vložení K-gramu se snižuje frekvence jeho složek $(K-1)$-gramů, $(K-2)$-gramů .... Jestliže se díky snižování frekvencí frekvence nějaké položky velmi sníží, je ze slovníku vyloučena.

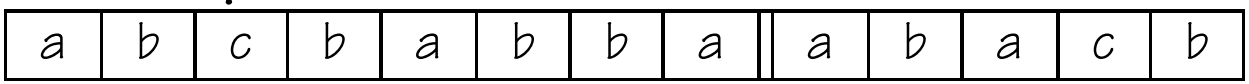
Adaptivní slovníkové metody

Lempel, Ziv 1977, 78

LZ77 – metody posuvného okna

LZ78 – metody rostoucího slovníku

Metoda posuvného okna



zakódovaná část (okno, $N \leq 8192$)

nezakód. část ($|B|$ ~10–20 b)

Krok kódování LZ77

V zakódované části je vyhledána nejdelší předpona P řetězu v nezakódované oblasti. Pokud je takový řetěz S nalezen, pak P je zakódováno pomocí (I, J, A), kde I je vzdálenost prvního znaku S od hranice, J je délka řetězu S a A je první znak za předponou P. Okno je posunuto o $J + 1$ znaků doprava. Jestliže podřetěz S nalezen nebyl, pak je vytvořena trojice $(0, 0, A)$, kde A je první znak nezakódované části.

LZR (Rodeh)

$|M| = (N - B) \times B \times t$, t velikost abecedy
$L(m) = \lceil \log_2(N - B) \rceil + \lceil \log_2 B \rceil + \lceil \log_2 t \rceil$
Výhoda: hledání nejdelší předpony [KMP]

LZR (Rodeh)

- LZR používá strom obsahující všechny předpony v dosud zakódované části.
- Je použita celá dosud zakódovaná část jako slovník.
- Protože i v (i, j, a) může být velké, je použit Eliasův kód pro zakódování celých čísel.

Nevýhoda: růst velikosti stromu bez omezení $\Rightarrow$ po překročení vymezené paměti vymazán a konstrukce začíná od začátku.

LZSS (Bell, Storer, Szymanski)

Kódem je posloupnost ukazatelů a znaků. Ukazatel (i, j) potřebuje paměť jak p znaků $\Rightarrow$ ukazatel jen tehdy, když ušetříme, ale je třeba bit na rozlišení znaku od ukazatele. Počet položek slovníku je $|M| = t + (N - B) \times (B - p)$ (uvažují se jen podřetězy delší než p). Počet bitů na zakódování je

- $L(m) = 1 + \lceil \log_2 t \rceil$ pro $m \in T$
- $L(m) = 1 + \lceil \log_2 N \rceil + \lceil \log_2 (B - p) \rceil$ jinak.

(Délku d podřetězu můžeme reprezentovat jako $B - p$).

LZB (Bell), LZH (Brent)

LZB (Bell)

Ukazatel (i, j) (analogie LZSS)

Jestliže

- okno není plné (na začátku) a
- komprimovaný text je kratší než N,

je plýtvání použití $\log_2 N$ bytů na zakódování i. LZB používá fázování při bin. kód. – prefixový kód s rostoucím počtem bitů pro rostoucí hodnoty čísel. Pro kódování j používá LZB Eliasův kód γ.

LZH (Brent)

LZSS, kde pro kódování ukazatelů je použito Huffmanovo kódování (tj. dle rozložení jejich pravděpodobností $\Rightarrow$ 2 průchody)

Metody s rostoucím slovníkem

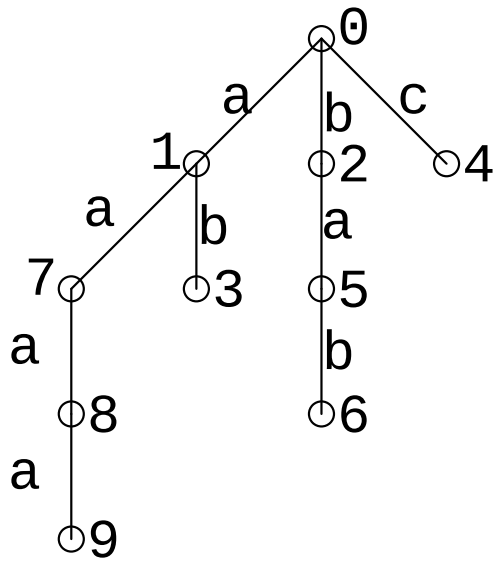
LZ78

Hlavní myšlenka: slovník obsahuje fráze. Nová fráze tak, že již existující fráze je rozšířena o symbol. Fráze je zakódována indexem předpony a přidaným symbolem.

LZ78 – příklad

Vstupní	a	b	ab	c	ba	...
Index	1	2	3	4	5	
Výstup	(0,a)	(0,b)	(1,b)	(0,c)	(2,a)	

...	Vstupní	bab	aa	aaa	aaaa
	Index	6	7	8	9
	Výstup	(5,b)	(1,a)	(7,a)	(8,a)



LZFG (Fiala, Green)

LZFG (Fiala, Green)

Slovník uložen ve stromové struktuře, hrany ohodnoceny řetězy znaků. Tyto řetězy jsou v okně a každý uzel stromu obsahuje ukazatel do okna a identifikující symboly na cestě z kořene do uzlu.

LZW (Welch), LZC

LZW

Výstupem jsou pouze indexy, nebo

- slovník je iniciován položkami pro všechny vstupní symboly,
- poslední symbol každé fráze je prvním symbolem následující fráze.

Vstup	a	b	a	b	c	b	a	b	a	b	a	a	a	a	a
Index		4	5		6	7		8			9	10			
Výstup	1	2		4	3		5			8	1		10	11	

Přeplnění $\Rightarrow$ další fráze není předávána a kódování pokračuje staticky.

LZC *compress*

je to LZW +

- Ukazatele jsou kódovány s prodlužující se délkou.
- Jakmile se kompresní poměr začne snižovat, slovník se vymaže a začíná se od začátku.

LZT, LZMW, LZJ

LZT (Tischer)

Jako LZC, ale při přeplnění slovníku se ze slovníku vylučují fráze, které byly v nedávné minulosti nejméně použity. Používá frázování při bin. kód. indexů frází.

LZMW (Miller, Wegman)

Jako LZT, ale nová fráze se nevytváří přidáním jednoho symbolu k předchozí, ale konstruuje novou frázi zřetězením dvou posledně zakódovaných.

LZJ (Jakobsson)

Jiný princip konstrukce slovníku:

- Na začátku vloženy jednotlivé symboly.
- Slovník uložen ve stromu a obsahuje všechny podřetězy zprac. řetězem do délky h.
- Plný slovník $\Rightarrow$
  - statický postup,
  - vynechávání uzlů s nízkou frekvencí použití.

Osnova (Týden jedenáctý)

- Slovníkové metody s restrukturalizací slovníku.
- Syntaktické metody.
- Kontrola správnosti textu.
- Dotazování a modely TIS.
- Booleovský model dokumentů.
- Vektorový model dokumentů.
- Architektura TIS.
- Signaturové metody.
- Podobnost dokumentů.

Slovníkové metody s restrukturalizací slovníku

- Průběžné uspořádávávání zdrojových jednotek ⟶ kratší řetězce kódu.
- Varianty heuristik (četnost, přesun na začátek (BSTW), výměna s předcházejícím, přesun o K vpřed).
- BSTW (výhoda: vysoká lokalita výskytů menšího počtu zdrojových jednotek).
- Příklad: Já do lesa nepojedu, ..., $1^n 2^n k^n$.
- Zobecnění: ***koeficient nedávnosti***, ***Intervalové kódování***.

Intervalové kódování

Reprezentace slova celkovým počtem slov od posledního výskytu. Slovník obsahuje slova $a_1, a_2, \ldots, a_n$, vstupní posloupnost $x_1, x_2, \ldots, x_m$. Hodnota LAST(a_i) obsahující interval od posledního výskytu je inicializovaná na nulu.

for $t := 1$ to m **do**
begin $\{x_t = a_i\}$
    if LAST($x_t = 0$) **then** $y(t) = t + i - 1$
        else $y(t) = t - \text{LAST}(x_t)$;
    LAST(x_t):=t
end .

Posloupnost $y_1, y_2, \ldots, y_m$ je výstupem kodéru a může být zakódována některým kódem proměnné délky.

Syntaktické metody

Derivační kódování

- je známa gramatika jazyka zprávy.
- levý rozklad derivačního stromu řetězce.
- globální číslování pravidel.
- lokální číslování pravidel.

Analytické kódování

- jsou kódovány rozhodovací stavy LR analyzátoru.

Kontextové modelování

- pevný kontext – model řádu N.
- kombinovaný přístup – kontexty různých délek.
- $p(x) = \sum_{n=0}^{m} w_n p_n(x)$.
- w_n pevné, proměnné.
- náročné na čas a paměť.
- přiřazení pravděpodobnosti nové zdrojové jednotce: $e = \frac{1}{c_n+1}$.
- automaty s konečným kontextem.
- dynamické Markovovo modelování.

Kontrola správnosti textu

- Kontrola textu pomocí frekvenčního slovníku.
- Kontrola textu pomocí *dvojitého* slovníku.
- Interaktivní kontrola textu (ispell).
- Kontrola textu založená na pravidelnosti slov, ***koeficient podivnosti***.

Koeficient podivnosti

Koeficient podivnosti trigramu xyz
$KPT = [\log(f(xy) - 1) + \log(f(yz) - 1)]/2 - \log(f(xyz) - 1)$, kde $f(xy)$ resp. $f(xyz)$ jsou relativní frekvence digramu resp. trigramu, $\log(0)$ je definován jako -10.

Koeficient podivnosti slova $KPS = \sqrt{\sum_{i=1}^{n}(KPT_i - SKPT^2)}$, kde KPT_i je koeficient podivnosti i-tého trigramu $SKPT$ je střední hodnota koeficientu podivnosti všech trigramů obsažených ve slově.

Dotazování a modely TIS

Různé metody hierarchizace a uložení dokumentů ⟶ různé možnosti a efektivita dotazování.

- Booleovský model, SQL.
- Vektorový model.
- Rozšířené booleovské modely.
- Pravděpodobnostní model.
- Model shluků dokumentů.

Blairovo ladění dotazu

Vyhledávání spočívá ve zmenšování neurčitosti tazatele (ladění dotazu).

1. Najdeme dokument s vysokou relevancí.
2. Začneme se dotazovat s jeho klíčovými slovy.
3. Odstraňujeme deskriptory, resp. je nahrazujeme disjunkcemi.

Infomap – snaha o sémantické dotazování

Systém **http://infomap.stanford.edu** – pro práci s hledaným významem/konceptem (na rozdíl od pouhých řetězců znaků). Správný dotaz je polovina odpovědi. Vyhledávání spočívá v určení sémanticky nejbližších termů.

Booleovský model

- 50. léta: reprezentace dokumentů pomocí množin termů a dotazování založené na vyhodnocování booleovských výrazů.
- Výraz dotazu: induktivně z primitiv:
  - term
  - jméno_atributu = hodnota_atributu (porovnání)
  - jméno_funkce(term) (aplikace funkce)

  a dále pomocí závorek a logických spojek X and Y, X or Y, X xor Y, not Y.
- disjunktivní normální forma, konjunktivní normální forma
- proximitní operátory
- regulární výrazy
- použití tezauru

Jazyky pro vyhledávání – SQL

- booleovské operátory **and**, **or**, **xor**, **not**.
- poziční operátory **adj**, **(n) words**, **with**, **same**, **syn**.
- SQL rozšíření: operace/dotazy s využitím tezauru

BT(A)	Broader term
NT(A)	Narrower term
PT(A)	Preferred term
SYN(A)	Synonyma termu A
RT(A)	Related term
TT(A)	Top term

Dotazování – SQL příklady

```
ORACLE SQL*TEXTRETRIEVAL
SELECT specifikace_polozek
FROM specifikace_tabulek
WHERE polozka
CONTAINS textovy_vyraz
```

Příklad:

```
SELECT TITLE
FROM BOOK
WHERE ABSTRACT
CONTAINS 'TEXT' AND RT(RETRIEVAL)
'řetěz' 'řetěz'*  *'řetěz' 'ře?ěz'
'ře%těz' 'řetěza' (m,n) 'řetězb'
'víceslovná fráze' BT('řetěz',n)
BT('řetěz',*) NT('řetěz',n)
```

Dotazování – SQL příklady

Příklad:

```
SELECT JMENO
FROM ZAMESTNANEC
WHERE VZDELANI
CONTAINS  RT(UNIVERSITA)
AND JAZYKY
CONTAINS 'ANGLICTINA' AND 'NEMCINA'
AND PUBLIKACE
CONTAINS 'KNIHA' OR NT('KNIHA',*)
```

Stilesova technika/ asociační faktor

$$asoc(Q_A, Q_B) = \log_{10} \frac{(fN - AB - N/2)^2 N}{AB(N - A)(N - B)}$$

A – počet dokumentů „zasažených“ dotazem Q_A
B – počet dokumentů „zasažených“ dotazem Q_B (jehož relevanci počítáme)
f – počet dokumentů „zasažených“ oběma dotazy
N – celkový počet dokumentů v TIS
cutoff (relevantní/ nerelevantní)
clustering*/*hnízdění 1. generace, 2. generace, ...

Vektorový model

Vektorový model dokumentů: Nechť $a_1, \ldots, a_n$ termy, $D_1, \ldots, D_m$ dokumenty, a ***matice relevance*** $W = (w_{ij})$ typu m, n,

$$w_{ij} \in \langle 0, 1 \rangle \begin{cases} 0 & \text{není relevantní} \\ 1 & \text{je relevantní} \end{cases}$$

Dotaz $Q = (q_1, \ldots, q_n)$

- $S(Q, D_i) = \sum_i q_i w_{ij}$ ***koeficient podobnosti***
- $head(sort(S(Q, D_i)))$ odpověď

Osnova (Týden dvanáctý)

- Vektorový model dokumentů (dokončení).
- Pravděpodobnostní model.
- Model shluků dokumentů.
- Architektura TIS.
- Signaturové metody.
- Podobnost dokumentů.
- Komprese pomocí neuronových sítí.

Vektorový model: pros & cons

CONS: nebere v úvahu ?"a"? ?"nebo"?

PROS: možné vylepšení:

- normování vah
  - **Frekvence termu TF**
  - **Inverzní frekvence dokumentu** $IDF \equiv \log_2 \frac{m}{k}$
  - Rozlišení termů
- normování vah pro dokument: $\frac{TD}{\sqrt{\sum_j TD_j^3}}$
- normování vah pro dotaz: $\left(\frac{1}{2} \times \frac{\frac{1}{2}TF}{\max TF_i}\right) \times \log_2 \frac{m}{k}$

[POK, strany 85–113].

Automatické strukturování textů

- Vzájemné vazby mezi dokumenty v TIS.
- Encyklopedie (OSN, Funk and Wagnalls New Encyclopedia).

- [SBA]
  `http://columbus.cs.nott.ac.uk/compsci/epo/epodd/ep056gs`
- Google/CiteSeer: „automatic structuring of text files“

Signaturové metody

Vyhledávací metody IV. Předzpracování textu i vzorku (signaturové metody).

Signatury

- řetězené
- vrstvené

Dále [POK, strany 65–76], viz podklady [MAR].

Seznam materiálů u Marečka

[MAR] Materiály k předmětu PV030 ke zkopírování v knihkupectví Mareček.

1. Slidy přednášek předmětu, 4 slidy na list A4.
2. kopie [MEL].
3. kopie [POK].
4. článek [MEH].
5. materiály o Google [GOO] (plus české shrnutí).
6. kapitola 5.7 z [KOR] (podobnost).
7. [MeM].
8. ostatní (NLP, diagram s kompresními algoritmy,...).