Introduction to

Information Retrieval

فصل پنجم: فشرده سازی شاخص

علی قنبری سرخی

چرا فشرده سازی؟ (در حالت کلی)

- فضای دیسک کمتر (کاهش هزینه ذخیره سازی و صرفه جویی در هزینه)
- ذخیره بیشتر داده در حافظه (افزایش سرعت)
- سرعت انتقال داده از دیسک به حافظه را افزایش می دهد (دوباره، سرعت را افزایش می دهد)
  - خواندن داده های فشرده و decompress در حافظه سریع تر از خواندن داده های غیر فشرده می‌باشد
  - پیش فرض: الگوریتم های Decompression سریعتر هستند.

چرا فشرده سازی در بازیابی اطلاعات استفاده می‌شود؟

- اول، ما فضا را برای لغت‌نامه در نظر می‌گیریم
  - انگیزه اصلی برای فشرده سازی لغت‌نامه: لغت‌نامه به اندازه کافی در حافظه اصلی کوچک نگه داشته شود.
- سپس برای فایل posting
  - انگیزش: فضای دیسک مورد نیاز را کاهش داده شود. (زمان لازم برای خواندن از دیسک را کاهش دهید)
- توجه: موتورهای جستجوی بزرگ بخش قابل توجهی از پست‌ها را در حافظه نگه می‌دارند.
- ما طرح‌های فشرده‌سازی مختلف برای فرهنگ لغت و پست‌ها را طراحی خواهیم کرد.

فشرده سازی Lossy در مقابل فشره سازی lossless

- فشرده سازی Lossy: برخی از اطلاعات نادیده گرفته می شود.
  - اغلب به صورت متناوب چندین مرحله پیش‌پردازش استفاده می‌شود.
- downcasing, stop words, porter, number elimination

- فشرده سازی lossless: تمام اطلاعات حفظ می‌شود.
- آنچه ما بیشتر در فشرده سازی شاخص انجام می‌دهیم

فشرده سازی بی اتلاف در مقابل فشره سازی با اتلاف

- فشرده سازی با اتلاف : برخی از اطلاعات نادیده گرفته می شود.
  - اغلب به صورت متناوب چندین مرحله پیش‌پردازش استفاده می‌شود.
  - غیر حساس کردن به حروف کوچک و بزرگ، ریشه گیری و حذف کلمات توقف

- فشرده سازی بی اتلاف: تمام اطلاعات حفظ می‌شود.
- آنچه ما بیشتر در فشرده سازی شاخص انجام می‌دهیم

5

The Reuters collection :مجموعه‌ی مدل

symbol	statistics	value
N	documents	800,000
L	avg. # tokens per document	200
M	word types	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term (= word type)	7.5
T	non-positional postings	100,000,000

6

تاثیر پیش پردازش روی تعداد عبارات، پست های غیر موقعیتی و نشانه های Routers

size of	word types (term)			non-positional postings			positional postings (word tokens)		
	dictionary			non-positional index			positional index		
	size	Δ	cml..	size	Δ	cml..	size	Δ	cml..
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	-2%	-2%	100,680,242	-8%	-8%	179,158,204	-9%	-9%
case folding	391,523	-17%	-19%	96,969,056	-3%	-12%	179,158,204	-0%	-9%
30 stop w's	391,493	-0%	-19%	83,390,443	-14%	-24%	121,857,825	-31%	-38%
150 stop w's	391,373	-0%	-19%	67,001,847	-30%	-39%	94,516,599	-47%	-52%
stemming	322,383	-17%	-33%	63,812,300	-4%	-42%	94,516,599	-0%	-52%

- این جدول تعداد عبارات را برای سطوح مختلف پیش پردازش (ستون ۲) نشان می دهد.
- تعداد عبارات، فاکتور اصلی در تعیین اندازه ی لغت نامه است.
- تعداد پست های غیر موقعیتی (ستون ۳) نماینگر اندازه ی مورد انتظار شاخص غیر موقعیتی مجموعه است.
- اندازه مورد انتظار شاخص موقعیتی مرتبط با تعداد موقعیت هایی است که باید کدگذاری کند (ستون ۴)

7

واژگان عبارت چقدر بزرگ است؟

- چند کلمه متمایز وجود دارد؟
- آیا امکان تخمین کلمات مجزا در یک مجموعه می باشد؟
- نه واقعا: در یک مجموعه حداقل ۷۰۲۰ ≈ ۱۰۳۷ کلمه مختلف از طول ۲۰ وجود دارد.
- واژگان با حجم مجموعه رشد می‌کنند.

قانون Heaps: تخمین تعداد عبارات

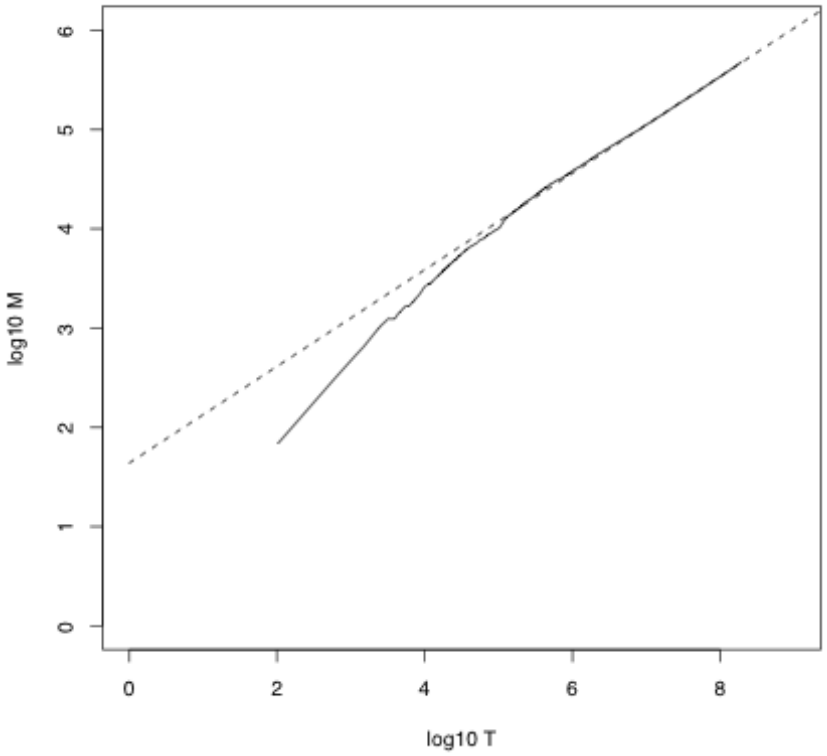
- این قانون، برای تعیین مقداد M استفاده می‌شود که اندازه ی مجموعه واژگان را به عنوان تابعی از اندازه ی مجموعه تخمین می زند.
- قانون Heaps:

$$M = kT^b$$

  - M اندازه واژگان است، T تعداد نشانه های مجموعه است.
  - مقادیر معمول برای پارامترهای k ($30 \leq k \leq 100$) و b ($b \approx 0.5$) است.
  - منشاء این قانون این است که ساده ترین رابطه ی ممکن بین اندازه ی مجموعه و اندازه مجموعه واژگان به صورت خطی در فضای log-log است.
- قانون تجربی

9

قانون Heaps برای Reuters



- اندازه ی مجموعه واژگان M به عنوان تابعی از اندازه ی مجموعه T (تعداد نشانه ها) برای Reuters-RCV1.
- برای این داده ها، خط چین $\log10M = 0.49 * \log10\ T + 1.64$ بهترین برازش کمترین مربعات است.
- بنابراین $k = 101.64 \approx 44$ و $b = 0.49$ است.

10

انطباق تجربی برای Reuters

- با توجه به گراف قبل برای Reuters
- برای 1,000,020 نشانه قانون Heap تعداد۳۸۳۲۳ عبارت را پیش بینی می کند.
- $44 \times 1{,}000{,}020^{0.49} \approx 38{,}323$
- تعداد واقعی ۳۸۳۶۵ عبارت است که بسیار به پیش بینی نزدیک است.
- قانون Heap بیان می‌کند:
  - اندازه لغت نامه، با افزایش تعداد اسناد در مجموعه افزایش می یابد، به جای اینکه یک حداکثر برای اندازه واژگان در نظر گرفته شود.
  - اندازه ی لغت نامه برای مجموعه های بزرگ، کاملا بزرگ است

11

Exercise

1. What is the effect of including spelling errors vs. automatically correcting spelling errors on Heaps' law?
2. Compute vocabulary size *M*
   - Looking at a collection of web pages, you find that there are 3000 different terms in the first 10,000 tokens and 30,000 different terms in the first 1,000,000 tokens.
   - Assume a search engine indexes a total of 20,000,000,000 (2×10^{10}) pages, containing 200 tokens on average
   - What is the size of the vocabulary of the indexed collection as predicted by Heaps' law?

قانون Zipf: مدلسازی توزیع عبارات

- ما می خواهیم بفهمیم که چگونه عبارات در طول اسناد توزیع می‌شوند.
- این امر به ما کمک می کند تا ویژگی های الگوریتم های فشرده سازی لیست پست ها را در بخش‌های بعدی مشخص کنیم.
- مدل رایج توزیع عبارات در یک مجموعه، قانون Zipf است.
- این قانون بیان می کند که اگر t1 رایج ترین عبارت در مجموعه، t2 عبارت رایج دوم باشد و به همین ترتیب، آنگاه فراوانی مجموعه iامین رایج ترین عبارت، cf_i با $1/i$ *متناسب است.* $\mathrm{cf}_i \propto \frac{1}{i}$
- *Cf تعداد اتفاق افتادن t در مجموعه می باشد.*

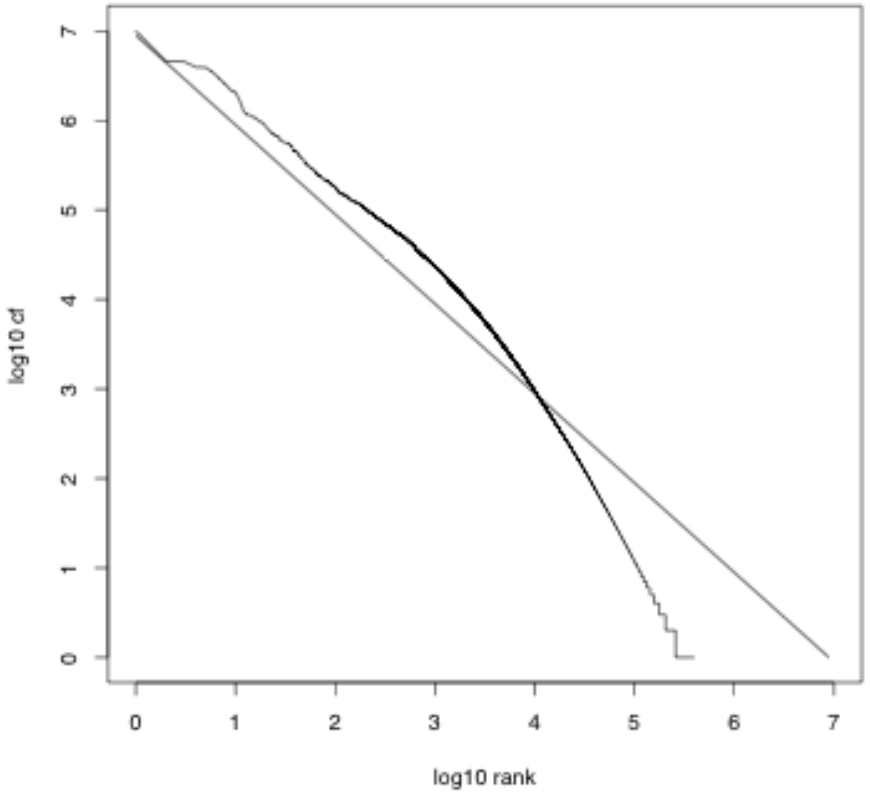
13

قانون Zipf: مدلسازی توزیع عبارات

- اگر عبارت با بیشترین فراوانی، cf_1 بار اتفاق بیفتد، پس دومین عبارت با فراوانی زیاد نیمی از وقوع‌های اولی را خواهد داشت. سومین عبارت با فراوانی زیاد یک سوم وقوع های اولی را خواهد داشت.
- مفهموم این است که نسبت به رتبه، بسیار سریع کاهش می یابد.
- رابطه $cf_i \propto \frac{1}{i}$ یکی از ساده ترین روش های فرموله کردن چنین کاهش سریعی است.
- همچنین مشخص شده است که این مدل بسیار معقول است.
- به صورت مشابه می توان این قانون را به صورت $cf_i = ci^k$ و $\log cf_i = \log c + k \log i$ نوشت.

14

قانون Zipf برای Reuters



- فراوانی به عنوان تابعی از رتبه ی فراوانی، برای عبارات در مجموعه رسم شده است.
- خط، توزیع پیش بینی شده توسط قانون Zipf است.
- برازش وزندار کمترین مربعات، تقطه تلاقی ۶.۹۵ است.

15

فشرده سازی لغت نامه

- لغت نامه در مقایسه با فایل پست ها کوچکتر است.
- چرا باید آن را فشرده کنیم در صورتیکه تنها درصد کمی از کل فضای لازم برای سیستم بازیابی اطلاعات را اشغال می‌کند؟
  - یکی از فاکتروهای اصلی در تعیین زمان پاسخگویی سیساتم بازیابی اطلاعات، تعداد پیگردهای مورد نیاز دیسک برای پردازش یک پرس و جو است.
  - اگر بخش‌های لغت نامه روی دیسک باشد پیگردهای دیسک بسیار بیشتری برای ارزیابی پرس و جو لازم است.
  - از اینرو هدف اصلی از فشرده سازی لغت نامه، گنجاندن آن در حافظه اصلی یا حداقل بخش اعظم آن است تا بازدهی بالایی برای پرس و جو فراهم شود.

16

یادآوری: لغت نامه به صورت آرایه ای با عرض ثابت

term	document frequency	pointer to postings list
a	656,265	⟶
aachen	65	⟶
. . .	. . .	. . .
zulu	221	⟶

Space needed: 20 bytes 4 bytes 4 bytes

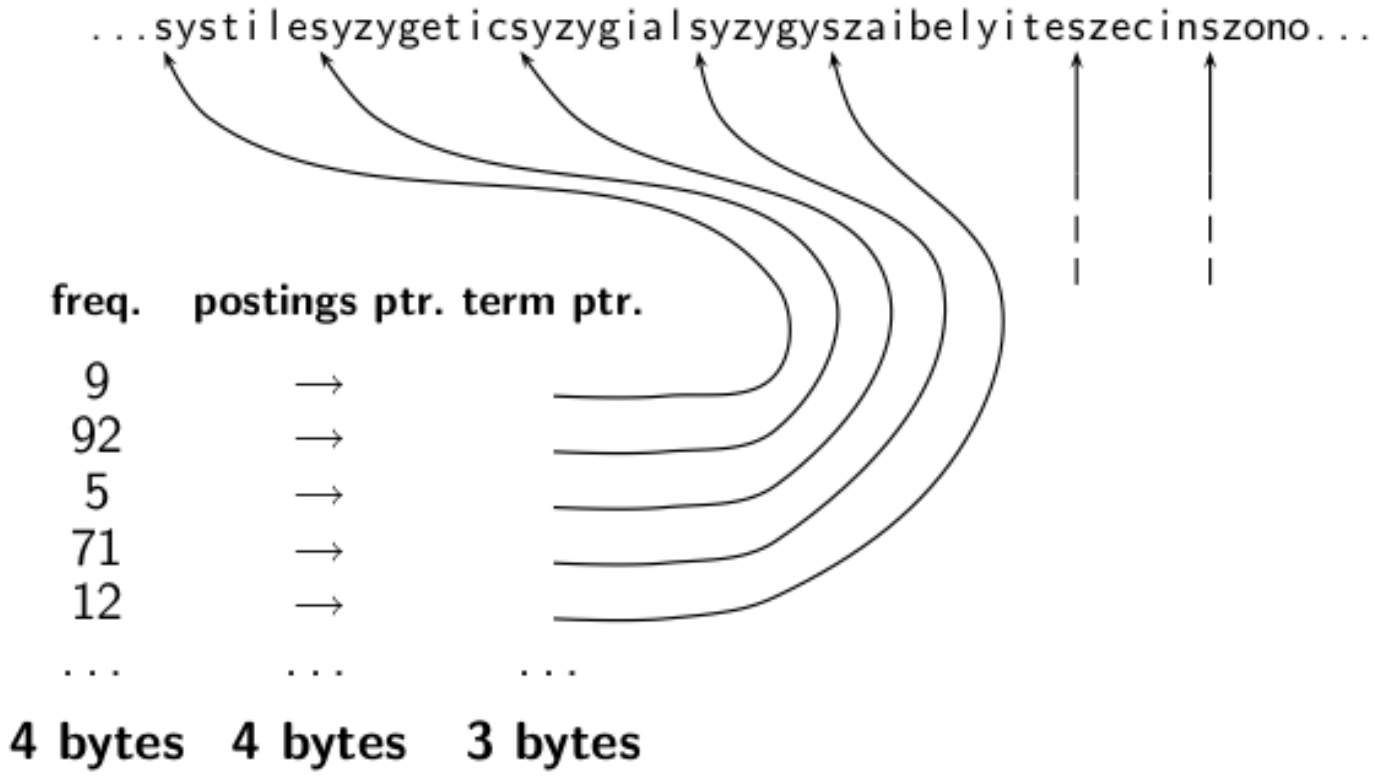
for Reuters: (20+4+4)*400,000 = 11.2 MB

- ساده ترین ساختمان داده برای لغت نامه: مجموعه واژگان را بر اساس لغات مرتب کنیم و آن را در یک آرایه از مدخلهایی با عرض ثابت ذخیره کنیم.

ورودی با عرض ثابت بی فایده است.

- میانگین طول هر عبارت در انگلیسی حداکثر ۸ کاراکتر است. بنابراین به طور متوسط، ۱۲ کاراکتر را در عرض ثابت هدر می دهیم.
- هیچ راهی برای ذخیره عبارات با بیش از ۲۰ کاراکتر مانند HYDROCHLOROFLUOROCARBONS و SUPERCALIFRAGILISTICEXPIALIDOCIOUS نداریم.
- مشکلاتی که در عرض ثابت مطرح شد را می توان با ذخیره عبارات لغت نامه به صورت یک **رشته** بلند از کاراکترها برطرف کرد.

لغت نامه به صورت یک رشته (String)



فضای لازم برای ذخیره لغت نامه به صورت رشته

- این روش در مقایسه با ذخیره با عرض ثابت تا ۶۰٪ کمتر به حافظه نیاز دارد.
- در این روش اشاره گر عبارات نیز باید ذخیره شود. اشاره گرهای عبارت $400000 \times 9 = 3.2$ $\times 10^6$ موقعیت را پوشش می دهند. بنابراین باید به طول $\log_2 3.2 \times 10^6 \approx 22$ بیت یا ۳ بایت باشند.
- در ذخیره به صورت رشته، ۴ بایت برای فراوانی، ۴ بایت برای اشاره گر پست ها، ۸ بایت (به صورت متوسط) برای هر عبارت در رشته و ۳ بایت اشاره گر عبارت لازم می باشد.
- در این طرح جدید $400 \times (4 + 4 + 3 + 8) = 7.6$ مگابایت برای لغت نامه در Reuters-RCV1 نیاز داریم. بنابراین فضای لازم را به اندازه یک سوم از ۱۱.۲ به ۷.۶ مگابایت کاهش دادیم.

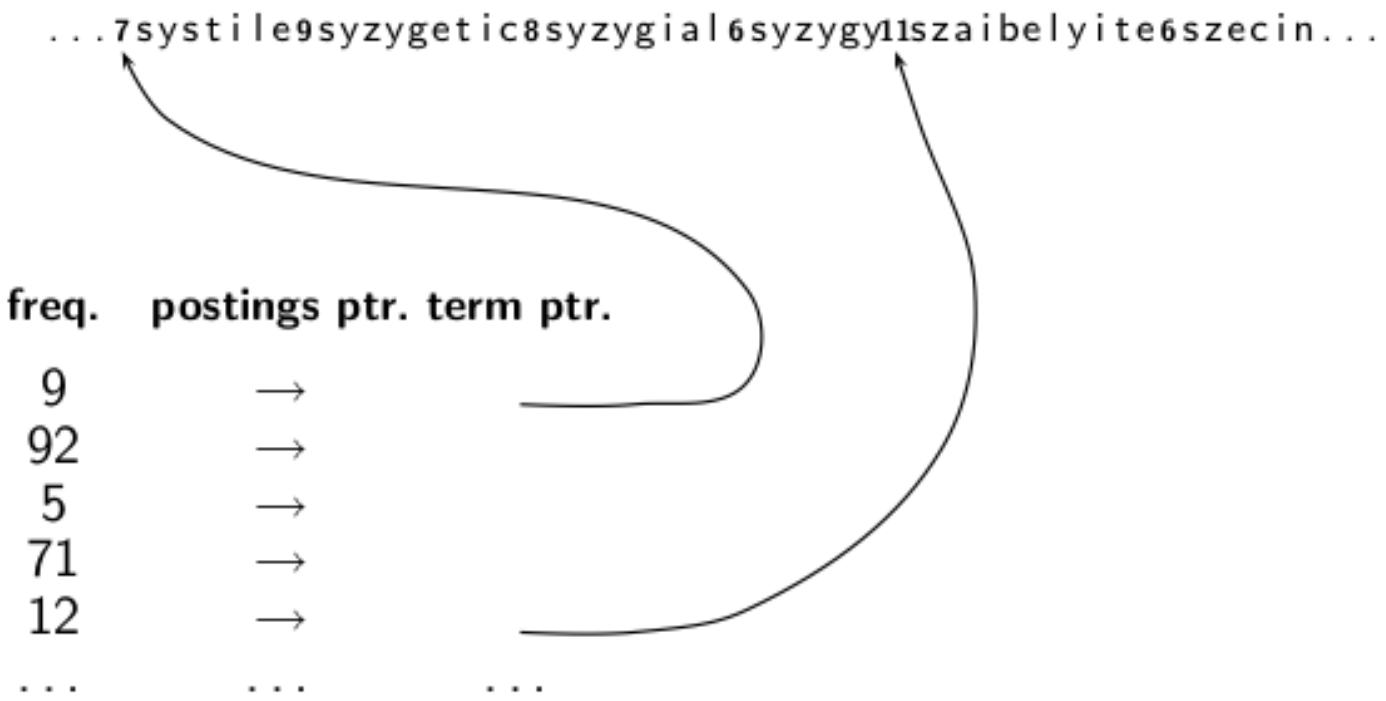
20

ذخیره سازی لغت نامه به صورت رشته با بلوکی

- لغت نامه با گروه بندی عبارات در رشته به بلوک های به اندازه k فشرده کرده و اشاره گر عبارت را تنها برای عبارت اول هر بلوک نگهداریم.
- طول عبارت در رشته را، به عنوان بایت اضافی در ابتدای هر عبارت ذخیره می کنیم. بنابراین اشاره گرهای k-1 عبارت را حذف می کنیم ، اما به یک k بایت اضاغی برای ذخیره طول هر عبارت نیاز داریم.
- برای مثال: برای k=4، ما $(k-1)\times 3 = 9$ بایت برای اشاره گر عبارات آزاد می کنیم، اما نیاز به k=4 بایت اضافه برای طول عبارات داریم.
- بنابراین کل فضای مورد نیاز برای لغت نامه تا ۵ بایت در هر بلوک ۴ عبارتی کاهش یافته یا در مجموع $400000 \times \frac{1}{4} \times 4 = 0.5$ مگابات، که تا ۷.۱ مگابایت فضا را کاهش می دهد.

21

ذخیره سازی لغت نامه به صورت رشته با بلوکی

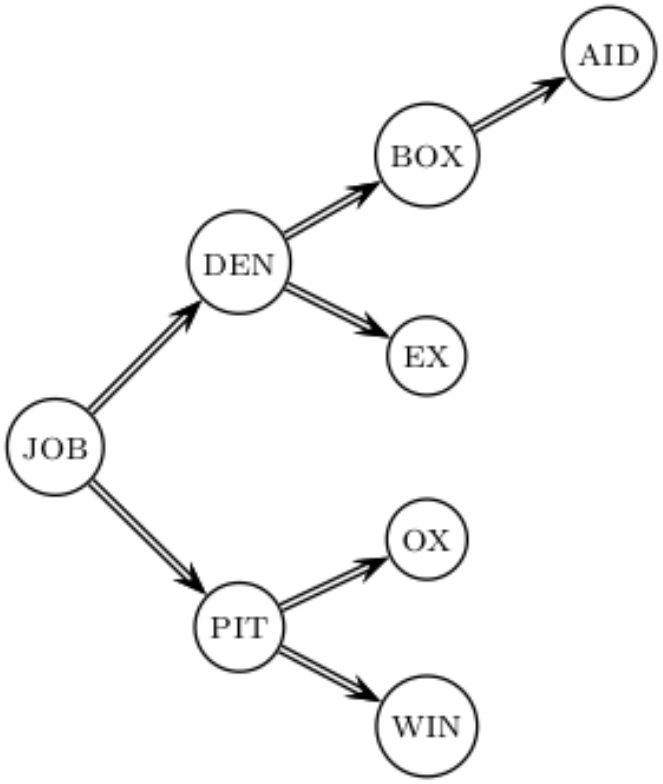


جستجو در لغت نامه به صورت رشته با بلوکی

- با افزای اندازه بلوک k، فشرده سازی بهتری خواهیم داشت. اگرچه، یک مصالحه بین فشرده سازی و سرعت جستجو عبارت وجو دارد.
- مقایسه جستجو (با توجه به شکل های دو اسلاید بعدی)
- برای لغت نامه ۸ عبارتی: گام ها در جستجو دودویی به صورت دو خطی و گام ها در جستجوی لیست به صورت خطوط ساده نشان داده شده است.
- لغت نامه فشرده نشده عبارت را با جستجو دودویی حستجو می کند.
- در لغت نامه فشرده شده ابتدا بلوک عبارت را با جستجوی دودویی تعیین موقعیت می کنیم و سپس موقعیت آن را در درون لیست با جستجو خطی در طول بلوک مشخص می کنیم.

23

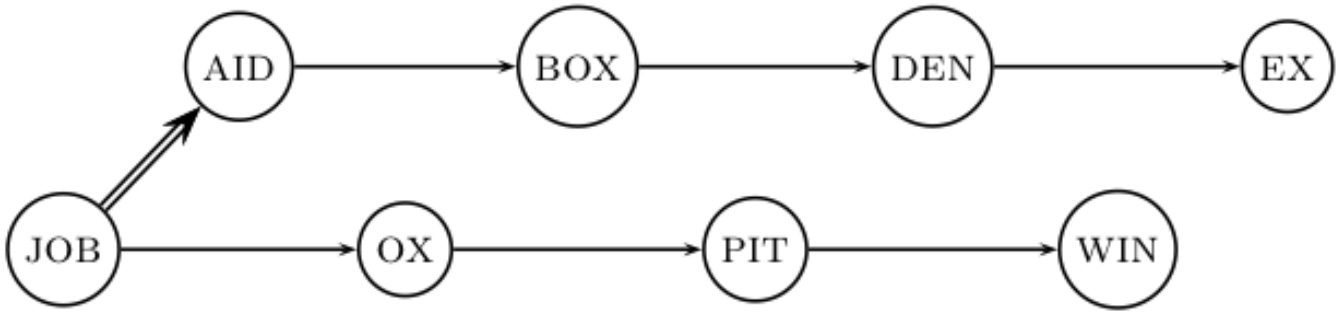
جستجو عبارت بدون بلوک بندی (فشرده نشده)



- جستجو لغت نامه فشرده نشده به طور متوسط $(0 + 1 + 2 + 3 + 2 + 1 + 2 + 2)/8 \approx 1.6$ گام دارد با فرض اینکه احتمال برخورد در یک پرس و جو برای هر عبارت مساوی باشد.
- برای مثال یافتن دو عبارت aid و box به ترتیب ۳ و دو گام را در بر میگیرد.

24

جستجو عبارت با بلوک بندی (کندتر)



- با بلوک های به اندازه k=4 ما به طور متوسط به $(0 + 1 + 2 + 3 + 4 + 1 + 2 + 2)/8 \approx 2$ گام که معادل ۲۵٪ بیشتر است نیاز دارد.
- برای مثال یافتن دو عبارت den یک گام جستجو دودویی و دو گام در طول بلوک نیاز دارد.

کدگذاری به طرف جلو

- یک منبع افزونگی در لغت نامه ای که هنوز به کار نگرفته ایم این است که مدخل های متوالی در یک لیست الفبایی پیشوندهای مشترک دارند. این شرایط به **کدگذاری به طرف جلو** منتهی می شود.
- یک پیشوند رایج برای زیردنباله ایاز لیست عبارت مشخص می شود و سپس با یک کاراکتر ویژه ارجاع داده می شود.
- آزمایشات نشان می دهد که کدگذاری به طرف جلو ۱.۲ مگابایت دیگر را آزاد می کند.

کدگذاری به طرف جلو

One block in blocked compression (k = 4) . . .

8 a u t o m a t a **8** a u t o m a t e **9** a u t o m a t i c **10** a u t o m a t i o n

⇓

. . . further compressed with front coding.

8 a u t o m a t ∗ a **1** ◊ e **2** ◊ i c **3** ◊ i o n

- کدگذاری به طرف جلو. یک دنباله از عبارات با پیشوند واحد (automat) توسط علامت گذاری انتهای پیشوند با ❋ و جایگزینی آن با ◊ در عبارت بعدی کدگذاری می شود.
- همانند گذشته، اولین بایت با مدخل، تعداد کاراکترها را کدگذاری می کند.

27

فشرده سازی لغت نامه برای Reuters-RCV1

data structure	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
∼, with blocking, k = 4	7.1
∼, with blocking & front coding	5.9

28

Exercise

- Which prefixes should be used for front coding? What are the tradeoffs?
- Input: list of terms (= the term vocabulary)
- Output: list of prefixes that will be used in front coding

29

فشرده سازی فایل پست ها

- فایل پست ها بسیار بزرگتر از لغت نامه می باشند.
- برای Reuters در ۸۰۰۰۰۰ سند، ۲۰۰ نشانه در هر سند، ۶ کاراکتر در هر نشانه و ۱۰۰۰۰۰۰۰۰ پست دارد.
- پست در این فصل به عنوان شناسه سند در یک لیست پست تعریف می شود. یعنی فراوانی و اطلاعات موقعیت را در نظر نمی گیریم.
- شناسه های سند به طول $\log_2 800{,}000 \approx 19.6 < 20$ بیت هستند.
- هدف اصلی: کمتر از ۲۰ بیت برای ذخیره سناسه سند استفاده شود.

30

ایده اصلی: ذخیره فواصل بجای شناسه سند

- هر لیست پست ها به ترتیب شناسه اسناد افزایش می یابد.
- برای نمونه، لیست پست‌ها: COMPUTER: 283154, 283159, 283202, . . .
- کافی است که فواصل را ذخیره کنیم: 283159-283154=5, 283202-283154=43
- نمونه از لیست پست ها با استفاده از فواصل: COMPUTER: 283154, 5, 43, . . .
- فواصل برای فراوانی عبارات کوچک هستند.
- بنابراین ما می توانیم فواصل کوچک را با کمتر از ۲۰ بیت کدگذاری کنیم.

31

کد گذاری فواصل

	encoding	postings list									
THE	docIDs	...		283042		283043		283044		283045	...
	gaps				1		1		1		...
COMPUTER	docIDs	...		283047		283154		283159		283202	...
	gaps				107		5		43		...
ARACHNOCENTRIC	docIDs	252000		500100							
	gaps	252000	248100								

- کدگذاری فواصل بجای شناسه های سند. برای مثال فواصل ۱۰۷، ۵، ۴۳ و ... را به جای شناسه های سند ۲۸۳۱۵۴، ۲۸۳۱۵۹، ۲۸۳۲۰۲، .... برای computer ذخیره می کنیم.
- اولین شناسه‌ی سند بدون تغییر می ماند (که تنها برای arachnocentric نشان داده شده است.)

32

کدگذاری طول متغیر

- ایده اصلی: فاصله بین پست ها کوتاه هستند و به فضای کمتر از ۲۰ بیت برای ذخیره شدن نیاز دارند.
- در حقیقت فاصله برای عیارات با فراوانی بیشتر مانند the و for برابر با ۱ است.
- اما برای عبارات نادرکه تنها یکی دو بار درمجموعه رخ می دهند (مانند ARACHNOCENTRIC) بزرگی به اندازه شناسه های سند دارد. . ۲۰ بیت نیاز دارد.
- برای نمایش به صرفه این توزیع فواصل، ما به روش کدگذاری متغیر نیاز داریم که بیت های کمتری را برای فواصل کوتاه به کار می برد.

کدهای بایت متغیر

- کدگذاری بایت متغیر: تعدادی صحیح زا بایت ها را برای کدگذاری یک فاصله به کار می برد.
- ۷ بیت آخر یک بایت، بار نامیده می شود و بخشی از فاصله را کدگذاری می کند.
- اولین بیت هر بایت، بیت ادامه نامیده می شود. این بیت برای بایت آخر فاصه کدگذاری شده، یکو در غیر این صورت صفر است.
- برای کدگشایی بایت متغیر، دنباله بایت هایی با بیت ادامه صفر را می خوانیم که با بایتی که بیت ادامه‌ی یک دارد پایان می یابد.
- سپس بخش های ۷ بیتی را جداکرده و الحاق می کنیم.

کدگذاری بایت متغیر

docIDs	824	829	215406	
gaps		5	214577	
VB code	00000110 10111000	10000101	00001101 00001100 10110001	

فواصل با استفاده از تعدادی صحیح از بایت ها کدگذاری می شوند. اولین بیت، بیت ادامه، از هر بایت نشان می‌دهد که آیا با این بایت ادامه می یابد (۱) یا ادامه نمی یابد (۰)

الگوریتم کدگذاری بایت متغیر

```
VBEncodeNumber(n)
1  bytes ← ⟨⟩
2  while true
3  do Prepend(bytes, n mod 128)
4     if n < 128
5        then Break
6     n ← n div 128
7  bytes[Length(bytes)] += 128
8  return bytes
```

```
VBEncode(numbers)
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBEncodeNumber(n)
4     bytestream ← Extend(bytestream, bytes)
5  return bytestream
```

الگوریتم کدگذاری بایت متغیر

```
VBDecode(bytestream)
 1  numbers ← ⟨⟩
 2  n ← 0
 3  for i ← 1 to Length(bytestream)
 4  do if bytestream[i] < 128
 5        then n ← 128 × n + bytestream[i]
 6        else n ← 128 × n + (bytestream[i] − 128)
 7             Append(numbers, n)
 8             n ← 0
 9  return numbers
```

کدهای متغیر دیگر

- ایده کدگذاری بایت متغیر را می توان برای واحدهای بزرگتر یا کوچکتر از بایت بکار برد.
- کلمات ۳۲ بیتی، کلمات ۱۶ بیتی و کلمات ۴ بیتی یا نیبل (Nibble)
- کلمات بزرگتر، میزان دستکاری بیت ها را به بهای فشرده سازی کمتر (با هیچ فشرده سازی) کاهش می دهند.
- اندازه کلمات کوچکتر از بایت ها حتی نسبت‌های فشرده سازی بهتری به بهای دستکاری بیشتر بیت‌ها ایجاد می کند.
- در حالت کلی، بایت‌ها سازش خوبی را بین فشرده سازی و سرعت از فشرده خارج کردن ایجاد می کنند.

کدهای گاما برای کدگذاری فواصل

- کدهای بایت متغیر، تعدادی بایت را بسته به اندازه‌ی فاصله به کار می برند. کدهای سطح بیت، طول کد را در سطح بیت به صورت دقیق‌تر تعیین می‌کند.
- ساده ترین کد سطح بیت، کد یگانی است. کد یگانی n، یک رشته n تایی از ۱ است که به دنبال آنها یک ۰ است. واضح است این کد چندان کارامد نیست اما بسیار در دسترس است.
- یک روش که نزدیک به فاکتور بهینه است کدگذاری گاما است. برای استفاده از این کدگذاری باید کدگذاری یگانی داشته باشیم . برای نمونه کدگذاری یگانی در زیر نشان داد
  - Represent *n* as *n* 1s with a final 0.
  - Unary code for 3 is 1110
  - Unary code for 40 is 1111111111111111111111111111111111111110
  - Unary code for 70 is:

11111111111111111111111111111111111111111111111111111111111111111110

39

کد گاما

- کدهای گاما، کدگذاری طول متغیر را با تقسیم نمایش فاصله G به زوج طول و آفست پیاده سازی می کند.
- آفست G دودویی است اما ۱ مقدم آن حذف شده است. برای نمونه: 13 → 1101 → 101 = offset
- طول، طول افست را در کدیگانی کدگذاری می کند. برای نمونه کد گاما برای ۱۳، 1110101 می‌شود که الحاق طول 1110 و آفست 101 است.
- یک کد گاما با خواند کد یگانی تا صفری که آن را تمام می کند، کدگشایی می‌شود.
- برای مثال چهار بیت 1110 زمانیکه 1110101 کدگشایی می شود. اکنون می دانیم که طول آفست چقدر است:۳ بیت
- آفست 101 می تواند به درستی خوانده شود و 1 که در کدگذاری جدا شده بود الحاق می شود: 101 → 1101=13

40

چند مثال از کدهای یگانی و کد گاما

number	unary code	length	offset	γ code
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110,0000000001

41

Exercise

- Compute the variable byte code of 130
- Compute the gamma code of 130

42

طول کد گاما

- طول آفست $\lfloor \log_2 G \rfloor$ بیت
- اندازه طول $\lfloor \log_2 G \rfloor + 1$ بیت
- بنابراین طول کل کد $2 \times \lfloor \log_2 G \rfloor + 1$ است.
- کدهای گاما همیشه طول فرد داشته و با فاکتور دو نسبت به طول کدگذاری بهینه‌ی $\log_2 G$ است.

43

ویژگی‌های کد گاما

- کدهای گام دو ویژگی مهم دارند که برای فشرده سازی شاخص مفید هستند:
- اولین ویژگی: پشوند آزاد هستند یعنی هیچ کد گاما پیشوند دیگری نیست. این بدان معنا می باشد که همیشه یک کدگشایی منحصر به فرد از دنباله کدهای گاما وجود دارد. و نیازی به جداکننده بین نیست.
- استفاده از جداکننده میان کدها ممکن است کارایی کد را کاهش دهد.
- دومی ویژگی: کدهای گام بدون پارامتر هستند. برای بسیاری از کدهای کارآمد دیگر، ما باید پارامترهای یک مدل را (مانند توزیع دو جمله ای) نسبت به توزیع فواصل تعیین کنیم.
- این امر پیاده سازی فشرده سازی و از حالت فشرده خارج کردن را پیچیده می کند.

44

Reuters فشرده سازی

data structure	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
∼, with blocking, k = 4	7.1
∼, with blocking & front coding	5.9
collection (text, xml markup etc)	3600.0
collection (text)	960.0
T/D incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, encoded	101.0

45