

# Tìm kiếm thông tin

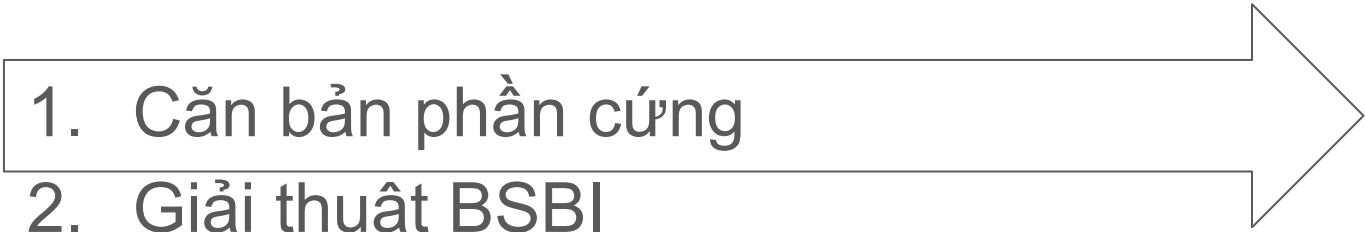
## Chương 3. Xây dựng và nén chỉ mục ngược

*Soạn bởi: TS. Nguyễn Bá Ngọc*

# Nội dung

1. Căn bản phần cứng
2. Giải thuật BSBI
3. Giải thuật SPIMI
4. Giải thuật dựa trên MapReduce
5. Chỉ mục ngược thay đổi động
6. Các đặc điểm thống kê từ vựng
7. Nén chỉ mục ngược

# Nội dung

- 
1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các đặc điểm thống kê từ vựng
  7. Nén chỉ mục ngược

# Phần cứng căn bản

- Tốc độ trao đổi dữ liệu (đọc/ghi) với bộ nhớ RAM nhanh hơn nhiều so với ổ đĩa;
- Không thể trao đổi dữ liệu với ổ đĩa khi đang định vị đầu đọc;
- Thời gian đọc/ghi nguyên một khối dữ liệu (block) hoặc một lượng nhỏ hơn là như nhau;
  - Kích thước khối được xác định trong quá trình định dạng ổ đĩa, phổ biến là 8, 16, 32, 64 Kb.
- BUS hệ thống điều khiển trao đổi dữ liệu giữa ổ đĩa và bộ nhớ RAM
  - Có thể sử dụng CPU trong thời gian này.

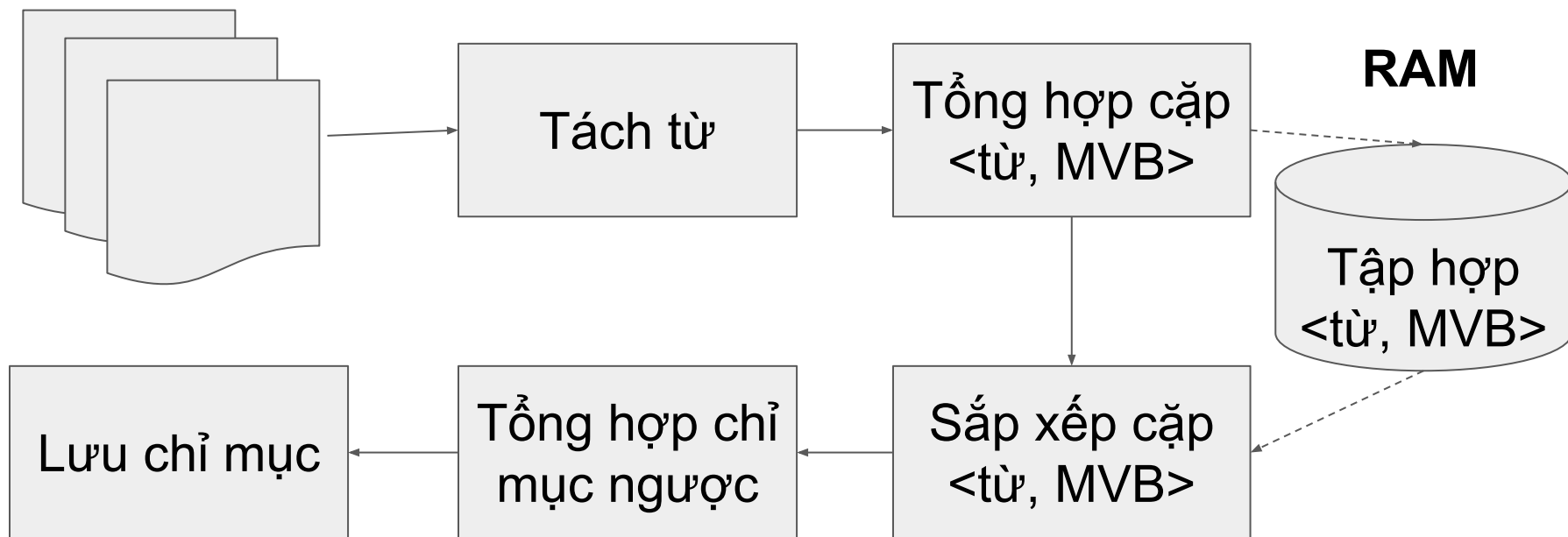
# Phần cứng căn bản (2)

- Các máy chủ được sử dụng cho các hệ thống TKTT hiện có vài chục GB cho tới hàng trăm GB RAM.
- Dung lượng ổ đĩa thường lớn hơn (2-3) cấp so với bộ nhớ chính (RAM).
- Phần cứng an toàn lỗi (hệ số tin cậy cao) có giá thành cao: Sử dụng phần cứng phổ thông rẻ hơn nhiều so với phần cứng an toàn lỗi.

# Các giả thuyết cho bài giảng

Ký hiệu	Đặc trưng	Giá trị
s	Thời gian định vị đầu đọc	5 ms = $5 \times 10^{-3}$ s
b	Thời gian trung bình đọc/ghi 1 byte	0.02 $\mu$ s = $2 \times 10^{-8}$ s
	Chu kỳ đồng hồ bộ vi xử lý	$10^{-9}$ s <sup>-1</sup>
p	Thời gian thực hiện một lệnh cơ bản	0.01 $\mu$ s = $10^{-8}$ s
	Dung lượng bộ nhớ chính	vài GB
	Dung lượng ổ đĩa	1 TB hoặc hơn

# (Ôn lại) Các bước xây dựng chỉ mục



# Mở rộng giải thuật xây dựng chỉ mục

- Khó mở rộng giải thuật xây dựng chỉ mục trong RAM cho những bộ dữ liệu lớn.
  - Kích thước bộ dữ liệu phụ thuộc vào dung lượng RAM
  - RAM có chi phí cao và hạn chế về dung lượng.
- Làm sao để tạo chỉ mục cho những bộ dữ liệu có kích thước lớn hơn dung lượng RAM?
  - Tính đến các giới hạn phần cứng: Tốc độ đọc/ghi v.v..

*Cần sử dụng ổ đĩa cứng: Chi phí thấp hơn RAM, nhưng dung lượng lưu trữ lớn.*



# Sử dụng ổ đĩa thay RAM để sắp xếp?

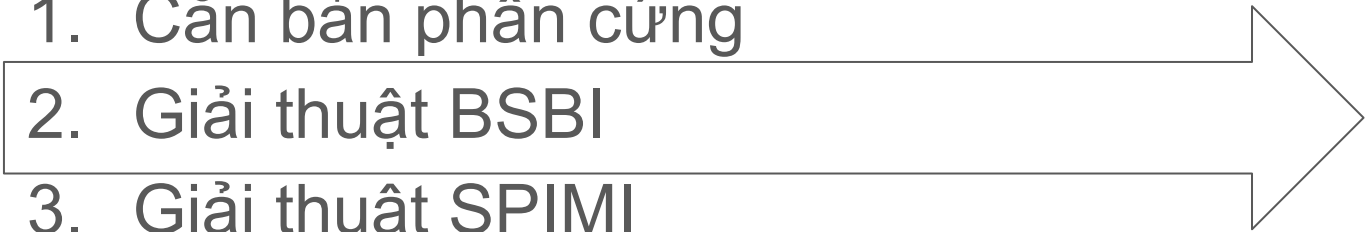
Liệu chúng ta có thể sử dụng cùng giải thuật tạo chỉ mục ngược cho bộ dữ liệu lớn hơn, nhưng sử dụng ổ đĩa thay cho RAM?

**Không!** Các thao tác truy cập ngẫu nhiên sẽ dẫn đến quá nhiều thao tác định vị đầu đọc => rất chậm

Thử ước lượng thời gian sắp xếp  $T = 100\,000$  bản ghi bằng giải thuật sắp xếp nhanh (qsort) với ổ đĩa cứng?

*Giải pháp: Sử dụng giải thuật sắp xếp ngoài*

# Nội dung

1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các đặc điểm thống kê từ vựng
  7. Nén chỉ mục ngược
- 

# Giải thuật BSBI

Tạo chỉ mục dựa trên sắp xếp khối: BSBI - Blocked Sort-Based Indexing

- Các thao tác cơ bản:
  - Đọc dữ liệu, tách từ và sinh cặp <từ, MVB>;
  - Tích lũy cặp <từ, MVB> thành khối kích thước giới hạn;
  - Sinh chỉ mục cho khối và lưu tạm thời trên ổ đĩa;
  - Hợp nhất các chỉ mục khối thành chỉ mục bộ dữ liệu.

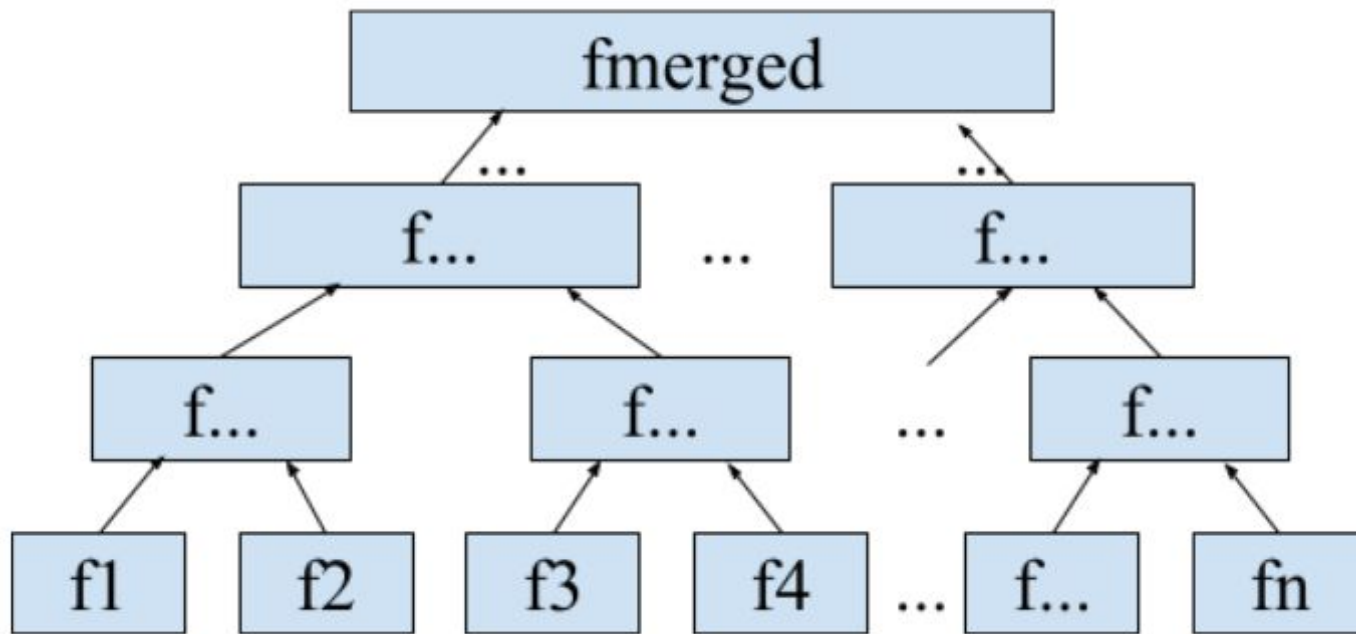
*Kích thước khối được lựa chọn đảm bảo có thể sắp xếp nhanh và tổng hợp danh sách thẻ định vị trong RAM;*

# Mã giả giải thuật BSBI

BSBIndexConstruction()

```
1  n = 0
2  while (khi chưa xử lý hết tất cả các văn bản)
3      n = n + 1
4      block = ParseNextBlock()
5      QSort(block)
6      BSBIInvert(block)
7      WriteBlockToDisk(block, fn)
8  MergeBlocks(f1, f2, ..., fn; fmerged)
```

# Sơ đồ cây hợp nhất n chỉ mục theo cặp



- Ý tưởng đơn giản, tuy nhiên dung lượng đọc ghi với ổ đĩa cứng lớn.
- => *Hợp nhất 1 lượt (đồng thời) n chỉ mục*

# Hợp nhất đồng thời n chỉ mục

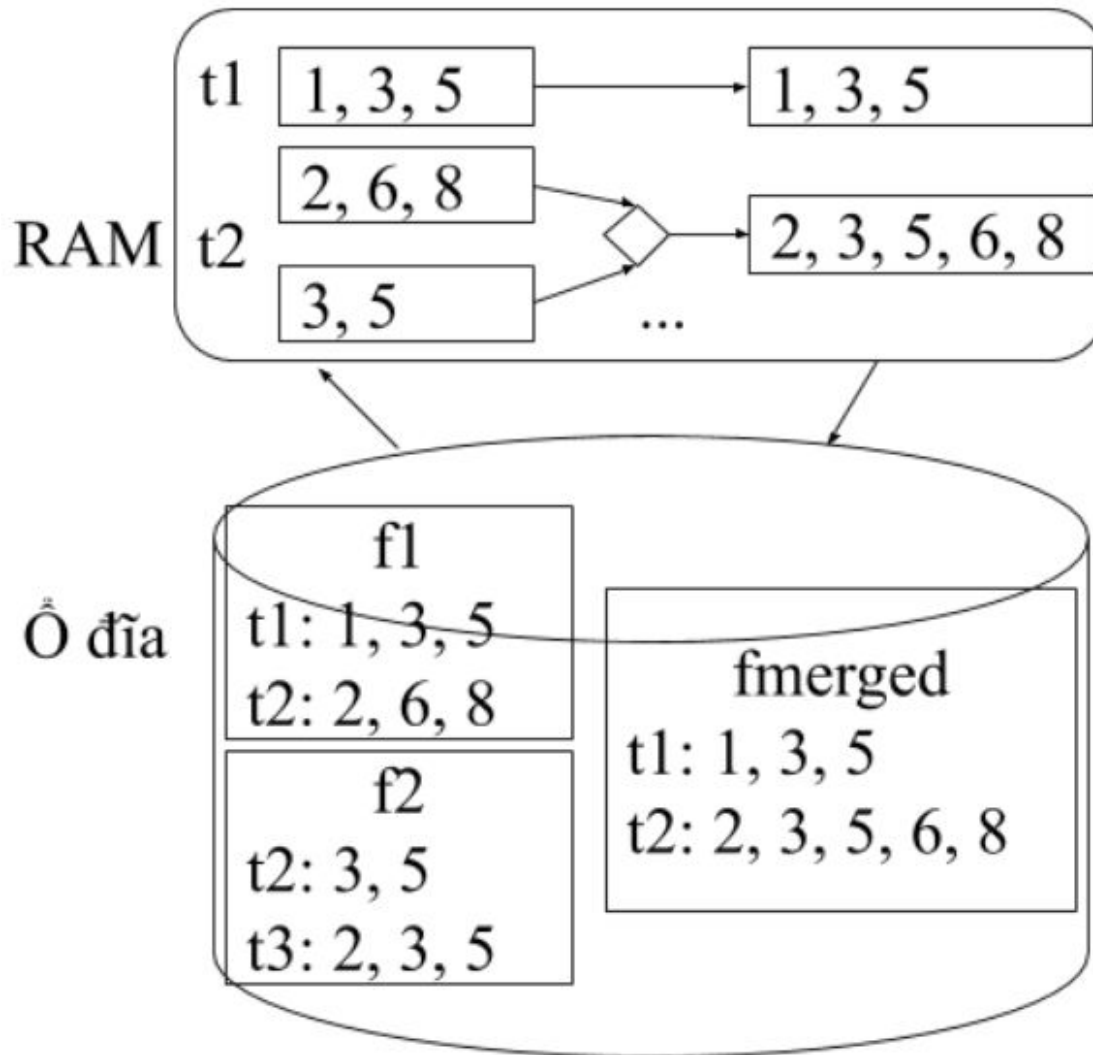
Mô tả giải thuật:

1. Sử dụng n vùng nhớ đệm cho n chỉ mục tạm thời.
2. Đọc dữ liệu từ các chỉ mục tạm vào các vùng nhớ đệm.
3. Lựa chọn từ nhỏ nhất và hợp nhất tất cả các danh sách thẻ định vị gắn với từ nhỏ nhất.
4. Lưu kết quả hợp nhất ra ổ đĩa.
5. Lặp các bước 2-4 cho tới khi đọc hết tất cả dữ liệu.

*Nếu có thể xử lý các văn bản theo thứ tự tăng dần mã số văn bản thì hợp nhất các danh sách thẻ định vị đơn giản = ghép nối danh sách thẻ định vị*

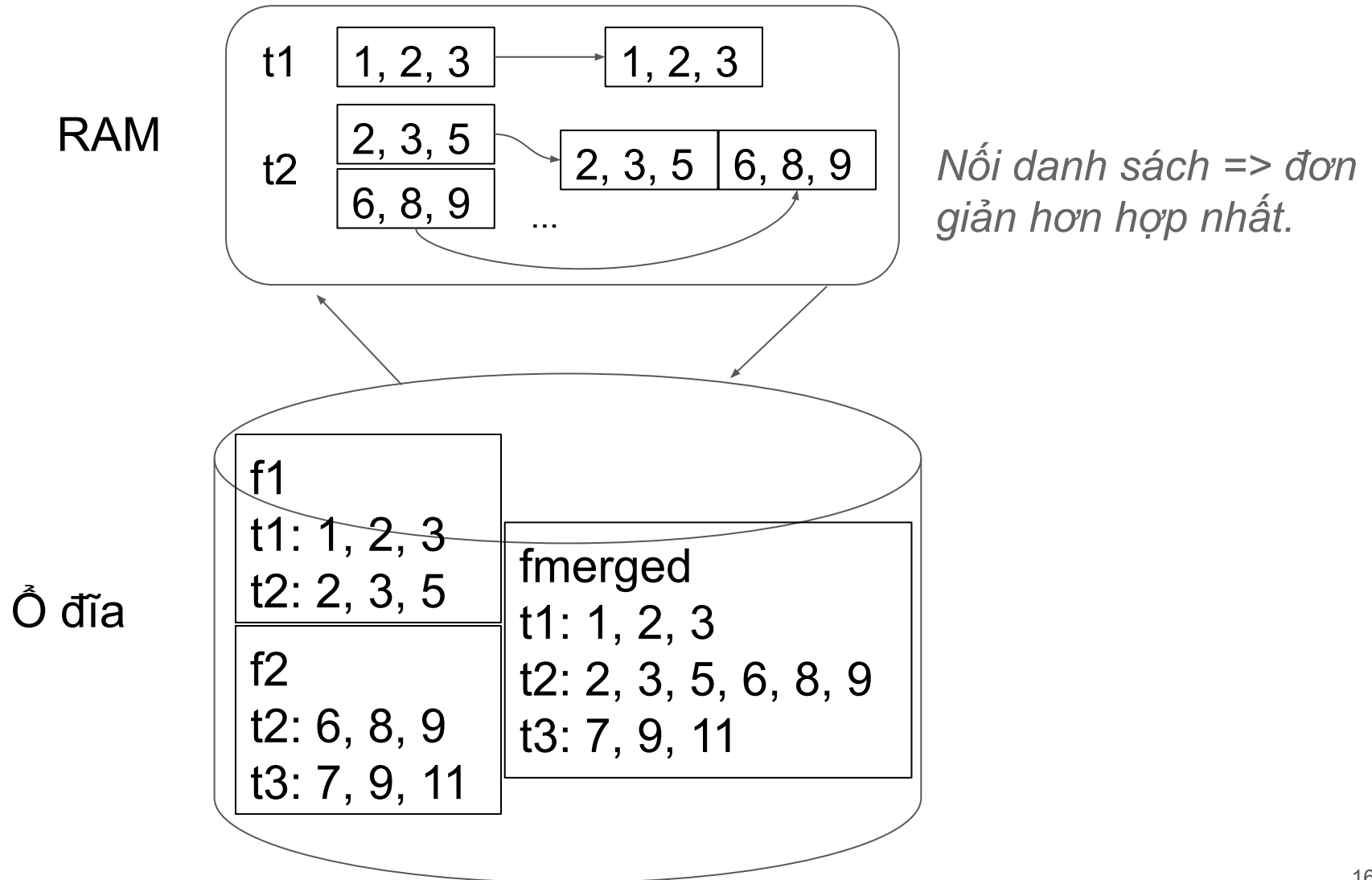
# Ví dụ 3.1.a. Hợp nhất các chỉ mục ngược

Trường hợp mã văn bản xen kẽ



# Ví dụ 3.1.b. Hợp nhất các chỉ mục ngược

Trường hợp mã văn bản tăng dần






# Mã từ

- Sử dụng từ điển để chuyển đổi từ thành mã từ (Nếu có thể):
  - Giả sử có thể lưu toàn bộ từ điển trong RAM
    - Giải thuật nén từ điển có thể giúp tăng số lượng từ trong bộ nhớ.
  - Sử dụng <Mã-từ, MVB> giúp giảm kích thước dữ liệu tạm thời => tăng tốc độ tạo chỉ mục

# Nội dung

1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các đặc điểm thống kê từ vựng
  7. Nén chỉ mục ngược
- 

# Giải thuật SPIMI

Tạo chỉ mục trong 1 lượt xử lý trong bộ nhớ: **Single-Pass In-Memory Indexing (SPIMI)**


- Sử dụng từ điển riêng cho từng khối;
  - Thường là bảng băm.
- Thêm trực tiếp thẻ định vị vào chỉ mục;
  - Vẫn đảm bảo trật tự tăng dần theo thứ tự xuất hiện.
- Kích thước chỉ mục tạm lớn hơn so với trong BSBI;
  - Giải thuật nén danh sách thẻ định vị có thể giúp tăng số lượng thẻ định vị trong bộ nhớ (*học sau*).
- Cần sắp xếp từ điển trước khi xuất ra ổ đĩa.

*Các chỉ mục tạm sau đó được hợp nhất thành một chỉ mục, tương tự như với BSBI*

# Mã giả giải thuật SPIMI

```
SPIMI-Invert(token_stream)
1  output_file = NewFile()
2  dictionary = NewHash()
3  while (bộ nhớ chưa đầy)
4      token = next(token_stream)
5      if term(token) không có trong bộ từ vựng
6          postings_list = AddToDictionary(dictionary, term(token))
7      else
8          postings_list = GetPostingsList(dictionary, term(token))
9      AddToPostingsList(postings_list, docID(term))
10 sorted_terms = SortTerms(dictionary)
11 WriteBlockToDisk(sorted_term, dictionary, output_file)
12 return output_file
```

# Nội dung

1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các đặc điểm thống kê từ vựng
  7. Nén chỉ mục ngược
- 

# Vấn đề tạo chỉ mục phân tán

- Khi kích thước dữ liệu vượt qua khả năng xử lý bằng một máy chủ
  - Ví dụ như dữ liệu Web
- Xử lý được thực hiện trong hệ thống tính toán phân tán sử dụng phần cứng phổ thông
  - Mỗi nút có thể gặp sự cố và ngưng hoạt động bất kỳ lúc nào.

*Làm thế nào đảm bảo hoàn thành xử lý trong hệ thống như vậy?*

# Trung tâm dữ liệu của máy tìm kiếm Web

- Các trung tâm dữ liệu của máy tìm kiếm Web (Google, Bing, Yandex) chủ yếu sử dụng các máy chủ phổ thông
  - Phân tán cả theo phương diện địa lý khắp nơi trên thế giới
- Ước lượng: Theo dữ liệu năm 2016 Google có khoảng 2.5 triệu máy chủ [Gartner].

# Các trung tâm dữ liệu quy mô lớn

- Nếu một hệ thống phân tán có 1000 máy chủ - các máy chủ đều sử dụng phần cứng phổ thông, nếu tỉ lệ thời gian hoạt động của mỗi nút là 99.9%, thì tỉ lệ thời gian hoạt động của toàn hệ thống là bao nhiêu?
- Trả lời: 37%/63%

**Thử:** Với cùng điều kiện, tính số lượng máy chủ gặp sự cố mỗi phút trong một hệ thống 1 triệu máy chủ.



# Phương pháp tạo chỉ mục phân tán

Một số ý tưởng cơ bản:

- Duy trì một nút điều hành (master) - điều khiển tiến trình tạo chỉ mục.
- Tập văn bản đầu vào được chia nhỏ thành nhiều khối (tương đương với các khối trong BSBI/SPIMI).
- Tiến trình tạo chỉ mục được chia nhỏ thành nhiều gói tính toán có thể được thực hiện song song.
- Nút điều hành giao các gói tính toán/giao việc cho các nút khác trong cụm (các workers) và quản lý tiến trình xử lý.
- Nếu một nút gặp sự cố trong quá trình xử lý thì gói tính toán đang được thực hiện ở đó được thu hồi và được giao cho nút khác.

*Các gói tính toán về mặt lô-gic có thể được chia thành 2 nhóm: Đọc dữ liệu và tổng hợp dữ liệu.*

# Đọc dữ liệu

- Nút điều hành gửi từng khối văn bản cho nút xử lý để tách từ và sinh thẻ định vị
  - Gói đọc dữ liệu.
  - Nút xử lý trong trường hợp này được gọi là nút đọc dữ liệu.
  - Đầu ra là các cặp <từ, MVB>
- Nút đọc dữ liệu lưu các cặp vào j tệp tương ứng với j khoảng từ
  - Có thể phân chia theo ký tự đầu tiên của từ
  - Ví dụ,  $j = 3 \Rightarrow$  3 khoảng từ: a-f, g-p, q-z

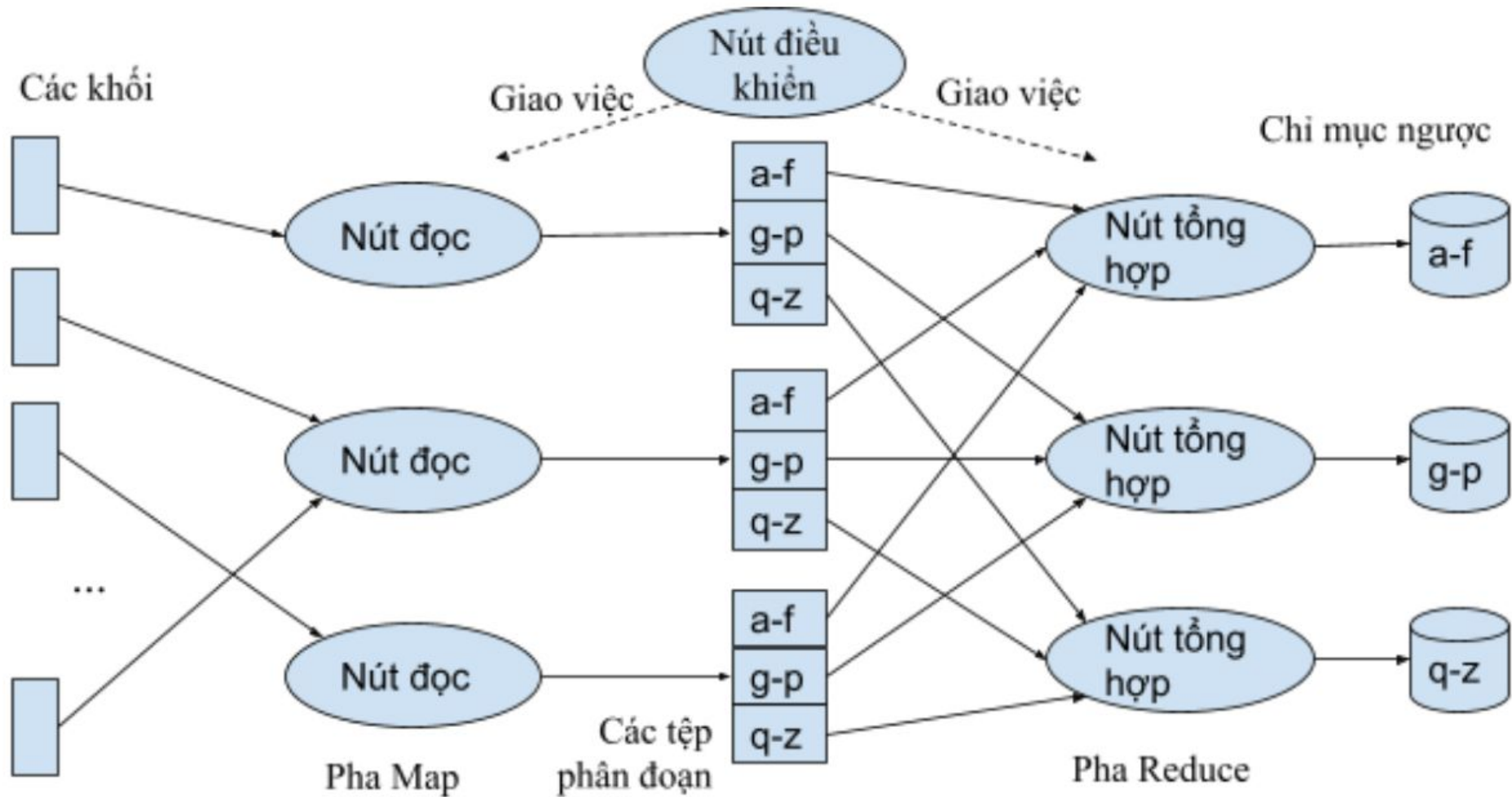
*Đữ liệu đầu ra của nút đọc sau đó được tổng hợp thành chỉ mục*

# Tổng hợp dữ liệu

- Nút điều hành gửi yêu cầu tổng hợp dữ liệu cho nút xử lý
  - Nút xử lý trong trường hợp này được gọi là nút tổng hợp
- Trong phạm vi một gói tổng hợp dữ liệu, mỗi nút tổng hợp gom tất cả các cặp <từ, MVB> tương ứng với một khoảng từ, thực hiện hợp nhất và tạo các danh sách thẻ định vị.

*Số lượng nút tổng hợp bằng số lượng phân đoạn từ (j)*

# Sơ đồ luồng dữ liệu



# MapReduce

- Phương pháp tạo chỉ mục chúng ta mới mô tả là một trường hợp của MapReduce
- MapReduce (Dean và Ghemawat 2004) là một nền tảng tính toán phân tán tin cậy và đơn giản ...
  - Không cần phải viết mã nguồn phân tán: Tương tác giữa các nút, phân chia công việc, trao đổi dữ liệu, v.v..
  - Đảm bảo hoàn thành công việc trên hệ thống tính toán phân tán được tạo thành từ phần cứng phổ thông
- Hệ thống đánh chỉ mục của Google (*hiện nay có thể khác*) được mô tả bao gồm nhiều pha, mỗi pha được triển khai theo MapReduce

# MapReduce<sub>(2)</sub>

- Lượt MapReduce đầu tiên nhận dữ liệu đầu vào là tập văn bản, và cho dữ liệu đầu ra là chỉ mục ngược phân tán theo từ.
  - Mỗi phần chỉ mục tương ứng với một khoảng từ.
- Hệ thống tìm kiếm còn có thể sử dụng thêm nhiều vòng MapReduce để biến đổi chỉ mục phân tán theo từ thành chỉ mục phân tán theo văn bản
  - Mỗi phần chỉ mục tương ứng với một khối văn bản => Được ưa dùng hơn.

*Thông tin chi tiết hơn được cung cấp trong phần tìm kiếm thông tin Web.*

# Định dạng hàm map và hàm reduce

- Khái quát

map: Dữ liệu đầu vào  $\Rightarrow$  list(key, value)

reduce: list(key, list(value))  $\Rightarrow$  Dữ liệu đầu ra

- Trong trường hợp tạo chỉ mục ngược

map: Tập văn bản  $\Rightarrow$  list(từ, MVB)

reduce: list(từ, list(MVB))  $\Rightarrow$  Chỉ mục ngược

## Ví dụ 3.2 Tạo chỉ mục theo MapReduce

- Pha Map:

d1 : C ca, C ce.

d2 : C d.

→

( $\langle C, d1 \rangle$ ,  $\langle ca, d1 \rangle$ ,  $\langle C, d1 \rangle$ ,  $\langle ce, d1 \rangle$ ,  $\langle C, d2 \rangle$ ,  $\langle d, d2 \rangle$ )

- Pha Reduce:

( $\langle C, (d1, d2, d1) \rangle$ ,  $\langle d, (d2) \rangle$ ,  $\langle ca, (d1) \rangle$ ,  $\langle ce, (d1) \rangle$ )


→

( $\langle C, (d1:2, d2:1) \rangle$ ,  $\langle d, (d2:1) \rangle$ ,  $\langle ca, (d1:1) \rangle$ ,  $\langle ce, (d1:1) \rangle$ )

*tf*



# Nội dung

1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các đặc điểm thống kê từ vựng
  7. Nén chỉ mục ngược
- 

# Cập nhật chỉ mục

- Khi có sự thay đổi với tập văn bản đầu vào thì chỉ mục cũng cần được cập nhật theo
  - Để có thể tìm kiếm được những văn bản xuất hiện sau khi tạo chỉ mục; ngừng trả về các văn bản đã bị xóa; v.v...
  - Những tập văn bản không bao giờ thay đổi (tập văn bản tĩnh) thường rất hiếm.
- Trong trường hợp tập văn bản có thể thay đổi nhưng không quá thường xuyên
  - Nếu thời gian tạo chỉ mục đủ nhỏ so với chu kỳ cập nhật, và thời gian trễ có thể chấp nhận được thì có thể cập nhật chỉ mục theo cách đơn giản là tạo lại chỉ mục.
    - Chỉ mục mới sẽ thay thế chỉ mục cũ.

*Đối với những bộ dữ liệu thay đổi thường xuyên cần phải có giải pháp quản lý hiệu quả hơn.*

# Các thao tác quản lý chỉ mục

- **Xóa (delete) văn bản:** Thường chỉ được giả lập vì thao tác này kéo theo những cập nhật phức tạp, có thể cần xây dựng lại chỉ mục.
  - Giả lập = Đánh dấu văn bản muốn xóa và lọc khỏi danh sách kết quả.
- **Thêm mới (insert):** Có nhiều phương pháp với độ phức tạp khác nhau để cho phép thêm mới văn bản vào chỉ mục (chi tiết được cung cấp sau).
- **Cập nhật (update):** Thường được thực hiện thông qua hai thao tác - xóa và thêm mới.

# Vấn đề cập nhật chỉ mục

- Trong thực tế các văn bản mới liên tục xuất hiện theo thời gian và cần được thêm vào chỉ mục, các văn bản đã có cũng có thể bị xóa hoặc được cập nhật.
- => Cần cập nhật chỉ mục:
  - Cập nhật danh sách thẻ định vị cho các từ đã có;
  - Thêm các từ mới vào từ điển.
  - V.V..

# Các chỉ mục chính&phụ

- Giải pháp tương đối đơn giản để quản lý các thay đổi với tập văn bản.
  - Duy trì một chỉ mục cho tất cả văn bản hiện có (chỉ mục chính, có kích thước lớn).
  - Đồng thời tạo một chỉ mục khác cho các văn bản mới được thêm vào (chỉ mục phụ, có kích thước nhỏ).
    - Khi số lượng văn bản mới được thêm vào chưa quá nhiều.
    - => có thể thường xuyên tạo lại chỉ mục phụ khi có văn bản mới.
    - Đến khi tập văn bản mới có kích thước đủ lớn & cần nhiều thời gian để tạo chỉ mục phụ => Cần đưa các thay đổi vào chỉ mục chính và làm rộng chỉ mục phụ.
- Tìm kiếm được thực hiện trên cả hai chỉ mục và sau đó cần hợp nhất các kết quả.
- Định kỳ tạo lại chỉ mục chính cho toàn bộ tập văn bản.

# Hạn chế của các chỉ mục chính và phụ

- Nếu các văn bản mới được thêm vào rất thường xuyên thì cần nhiều thời gian để tạo lại các chỉ mục:
  - Hợp nhất chỉ mục phụ với chỉ mục chính chỉ hiệu quả nếu chúng ta lưu mỗi danh sách thẻ định vị trong một tệp riêng
    - ... thao tác hợp nhất tương đương với thêm vào cuối tệp
    - Tuy nhiên sẽ cần nhiều tệp => bị giới hạn bởi khả năng quản lý tệp của HĐH
  - Trong thực tế: Các danh sách thẻ định vị có thể được lưu trong 1 tệp hoặc
  - ... chia nhỏ những danh sách thẻ định vị rất lớn và gom nhiều danh sách có độ dài ngắn trong một tệp
    - => Tốn khá nhiều thời gian để hợp nhất.

*Chúng ta cần giải pháp khác hiệu quả hơn nếu các văn bản thay đổi rất thường xuyên.*

# Tạo chỉ mục tăng dần

- Ý tưởng cơ bản: Có thể liên tục thêm các văn bản vào chỉ mục. Chỉ mục mới được tạo ra được hợp nhất với chỉ mục đã có sẵn với kích thước tương đương tạo thành chỉ mục có kích thước lớn gấp đôi.
- Trong kịch bản các văn bản liên tục được thêm vào:
  - Văn bản mới được thêm vào chỉ mục  $Z_0$  trong RAM;
  - Đến khi  $Z_0$  đạt được kích thước giới hạn thì thực hiện tiến trình hợp nhất với dữ liệu trên ổ đĩa sau đó làm rỗng  $Z_0$ .
  - Tiến trình hợp nhất được thực hiện như sau:
    - Trong lần đầu tiên xuất  $Z_0$ ,  $Z_0$  được lưu vào ổ đĩa thành  $I_0$ .
    - Trong lần xuất  $Z_0$  tiếp theo,  $Z_0$  được hợp nhất với  $I_0$  đã có trong ổ đĩa, tạo thành chỉ mục mới ký hiệu là  $I_1$ . Sau đó  $I_0$  được xóa khỏi ổ đĩa.
    - Trong lần tiếp theo,  $Z_0$  được lưu thành  $I_0$ . ( $I_1$  vẫn được giữ nguyên)
    - ...
    - Chúng ta có  $Z_0 \Rightarrow I_0$ ;  $Z_0 + I_0 \Rightarrow I_1$  (hoặc  $Z_1$ );  $Z_1 + I_1 \Rightarrow I_2$  (hoặc  $Z_2$ ); ...

# Mã giả giải thuật tạo chỉ mục tăng dần

LMERGEADDTOKEN(*indexes*,  $Z_0$ , *token*)

```
1   $Z_0 \leftarrow \text{MERGE}(Z_0, \{\text{token}\})$ 
2  if  $|Z_0| = n$ 
3      then for  $i \leftarrow 0$  to  $\infty$ 
4          do if  $l_i \in \text{indexes}$ 
5              then  $Z_{i+1} \leftarrow \text{MERGE}(l_i, Z_i)$ 
6                  ( $Z_{i+1}$  is a temporary index on disk.)
7                   $\text{indexes} \leftarrow \text{indexes} - \{l_i\}$ 
8              else  $l_i \leftarrow Z_i$     ( $Z_i$  becomes the permanent index  $l_i$ .)
9                   $\text{indexes} \leftarrow \text{indexes} \cup \{l_i\}$ 
10                 BREAK
11          $Z_0 \leftarrow \emptyset$ 
```

LOGARITHMICMERGE()

```
1   $Z_0 \leftarrow \emptyset$     ( $Z_0$  is the in-memory index.)
2   $\text{indexes} \leftarrow \emptyset$ 
3  while true
4  do LMERGEADDTOKEN(indexes,  $Z_0$ , GETNEXTTOKEN())
```



# Phân tích giải thuật

- Đối với trường hợp chỉ mục chính&phụ: Thời gian tạo chỉ mục là  $O(T^2)$  bởi vì mỗi lần hợp nhất đều xử lý tất cả các thẻ định vị.
- Trong trường hợp nhất với độ phức tạp Lô-ga-rit (tạo chỉ mục tăng dần) mỗi thẻ định vị được xử lý  $O(\log T)$  lần, vì vậy độ phức tạp là  $O(T \cdot \log T)$ .
  - $\Rightarrow$  Tạo chỉ mục tăng dần nhanh hơn.
- Truy nhiên với cấu trúc chỉ mục lô-ga-rit, xử lý truy vấn yêu cầu hợp nhất tất cả các kết quả từ  $O(\log T)$  chỉ mục.
  - Với trường hợp chỉ mục chính & phụ chúng ta chỉ cần hợp nhất các kết quả từ 2 chỉ mục ( $O(1)$ ).

# Một số vấn đề tiêu biểu


Sử dụng nhiều chỉ mục làm phát sinh một số vấn đề tiêu biểu như:

- Khó xử lý các đại lượng thống kê toàn cục;
  - Ví dụ, với vấn đề sửa lỗi viết: Chúng ta tổng hợp số lượng kết quả từ tất cả các chỉ mục? Hay chỉ sử dụng dữ liệu trong chỉ mục lớn nhất?
- Khi giả lập thao tác xóa (với các vec-tơ đánh dấu), các văn bản được coi như đã xóa nhưng vẫn ảnh hưởng đến các đại lượng thống kê
  - Hệ quả là có thể ảnh hưởng tới các xếp hạng;
- Các kết quả được xếp hạng trong phạm vi thống kê khác nhau (trong các chỉ mục khác nhau) vậy làm sao để lựa chọn kết quả tốt nhất khi hợp nhất?

# Tạo chỉ trong máy tìm kiếm

- Tất cả các máy tìm kiếm lớn đều có thể tạo chỉ mục động:
  - Chỉ mục tăng dần và thay đổi liên tục
  - Các nội dung mới: Các bản tin mới, bài viết trong diễn đàn, các trang web, v.v., liên tục xuất hiện
- Tuy nhiên các chỉ mục vẫn được tạo lại từ đầu theo chu kỳ:
  - Chỉ mục mới sẽ thay thế chỉ mục cũ
  - Hữu ích cho việc cập nhật các đại lượng thống kê
  - Làm rỗng các vec-tơ đánh dấu
  - V.v..

# Nội dung

1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các đặc điểm thống kê từ vựng
  7. Nén chỉ mục ngược
- 

# Kích thước từ điển so với số lượng từ đặc trưng

- Bộ từ vựng lớn tới đâu?
  - Hay diễn đạt theo cách khác bộ từ vựng có bao nhiêu từ?
- Kích thước bộ từ vựng là hữu hạn hay vô hạn?
  - Trong thực tế kích thước bộ từ vựng tăng lên cùng với kích thước tập văn bản
  - Đặc biệt với Unicode

# Luật Heap

*Được phát hiện bằng thực nghiệm*

- $M = kT^b$ , trong đó  $M$  là kích thước bộ từ vựng;  $T$  là số từ trong tập văn bản;  $k, b$  là các hằng số gắn với tập văn bản cụ thể.
  - Các giá trị thường gặp:  $30 \leq k \leq 100$  và  $b \approx 0.5$
- Quan hệ tuyến tính trong mặt phẳng log-log:

$$\log(M) = \log(k) + b \log(T)$$

Các đại lượng  $T$  và  $M$  tạo thành một đường thẳng với độ dốc  $= b$ .

# Luật Heap <sub>(2)</sub>

- Có thể dự đoán kích thước bộ từ vựng trong tiến trình tạo chỉ mục ngược bằng luật Heap
  - Giả sử luật Heap đúng với bộ dữ liệu đang được xử lý.
  - Lấy mẫu các đại lượng M và T trong quá trình tạo chỉ mục ngược, sau đó tính k và b
  - Sau khi dự đoán được T, chúng ta có thể suy ra được M

*Có thể tính k, b dựa trên các giá trị T, M thu được trong tiến trình tạo chỉ mục tương tự cách xác định đường thẳng với lỗi tối thiểu trên các dãy X và Y:*

$$b_1 = \frac{\text{cov}(X, Y)}{\text{var}(X)} \quad b_1 = \frac{\sum (X_i - \bar{X}) \cdot (Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2}$$
$$b_0 = \bar{Y} - b_1 \bar{X} \quad y = b_0 + b_1 x$$


# Luật Zipf

*Là một luật thống kê dựa trên thực nghiệm khác.*

- Cho biết quan hệ giữa số lần xuất hiện và mức độ phổ biến của từ.
- Từ thường xuyên thứ  $i$  có tần suất tỉ lệ với  $1/i$ 
  - Có rất ít từ được sử dụng rất thường xuyên và có rất nhiều từ hiếm
  - $cf_i \propto 1/i = K/i$ , trong đó  $K$  là một hằng số chuẩn hóa,  $cf_i$  là tần suất tập văn bản (số lần xuất hiện từ trong tập văn bản).
  - Tương đương với  $\log cf_i = \log K - \log i$
  - $\Rightarrow$  Mối quan hệ tuyến tính trong mặt phẳng log-log.



# Nội dung

1. Căn bản phần cứng
  2. Giải thuật BSBI
  3. Giải thuật SPIMI
  4. Giải thuật dựa trên MapReduce
  5. Chỉ mục ngược thay đổi động
  6. Các loại chỉ mục khác
  7. Nén chỉ mục ngược
- 

# Vì sao cần nén dữ liệu?

- Giảm dung lượng lưu trữ
  - Tiết kiệm kinh phí, tiết kiệm tài nguyên lưu trữ
- Lưu nhiều dữ liệu hơn trong RAM
  - Tăng tốc độ xử lý
- Tăng tốc độ truyền dữ liệu từ ổ đĩa vào bộ nhớ
  - [Đọc dữ liệu đã nén => Giải nén] có thể nhanh hơn đọc dữ liệu không nén.

# Các loại giải thuật nén

- Nén bảo toàn:
  - Dữ liệu được bảo toàn sau khi giải nén => Khôi phục được dữ liệu ban đầu
  - Được sử dụng phổ biến trong tìm kiếm.
- Nén không bảo toàn:
  - Loại bỏ một phần dữ liệu => Không khôi phục được dữ liệu ban đầu, tuy nhiên tỉ lệ nén thường cao hơn nén bảo toàn;
  - Có thể coi một số bước xử lý từ vựng trong tiến trình tạo chỉ mục như nén không bảo toàn: Chuyển về chữ thường, loại từ dừng, loại bỏ giá trị số v.v.
  - Cắt tỉa không gian tìm kiếm (chi tiết được cung cấp sau)

# Vì sao cần nén chỉ mục ngược?

- Giảm dung lượng lưu trữ:
  - Rút ngắn thời gian đọc từ ổ đĩa vào RAM;
  - Lưu nhiều dữ liệu chỉ mục hơn trong RAM.
- Có thể nén từ điển
  - Danh sách từ đã sắp xếp có tỉ lệ nén cao.
- Có thể nén các danh sách thẻ định vị
  - Danh sách mã văn bản là các số nguyên được sắp xếp theo thứ tự tăng dần có tỉ lệ nén cao.

Nén từ điển

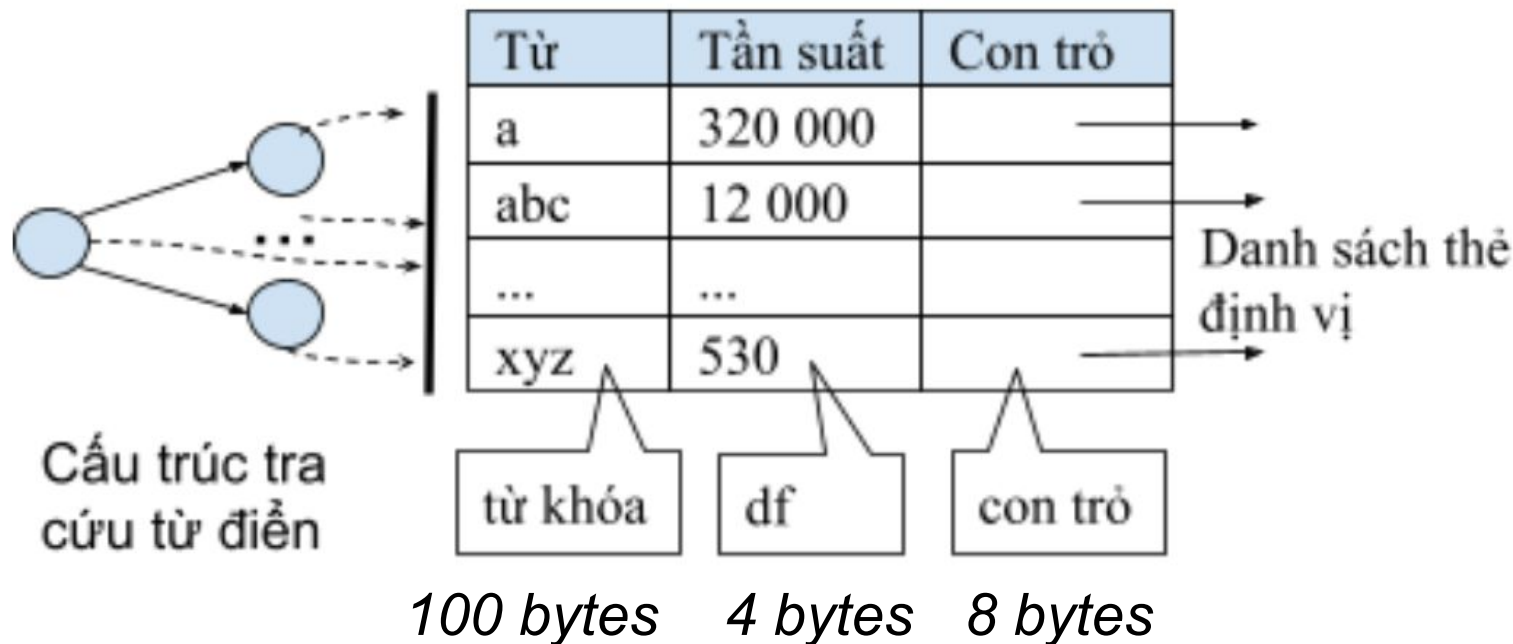
# Mục đích nén từ điển

- Lưu toàn bộ từ điển trong RAM:
  - Từ điển được truy cập rất thường xuyên.
  - Lưu từ điển trong RAM giúp xử lý nhanh.
  - Tuy nhiên RAM cũng cần được sử dụng cho các mục đích khác nữa.
  - => Nén từ điển có ý nghĩa quan trọng
    - Giảm dung lượng lưu trữ => Tiết kiệm RAM.
    - Rút ngắn thời gian đọc từ ổ đĩa cứng => Giảm thời gian khởi động.
- Có thể lưu danh sách từ với số lượng bytes nhỏ hơn tổng số lượng bytes của các từ trong danh sách.

# Biểu diễn cơ bản

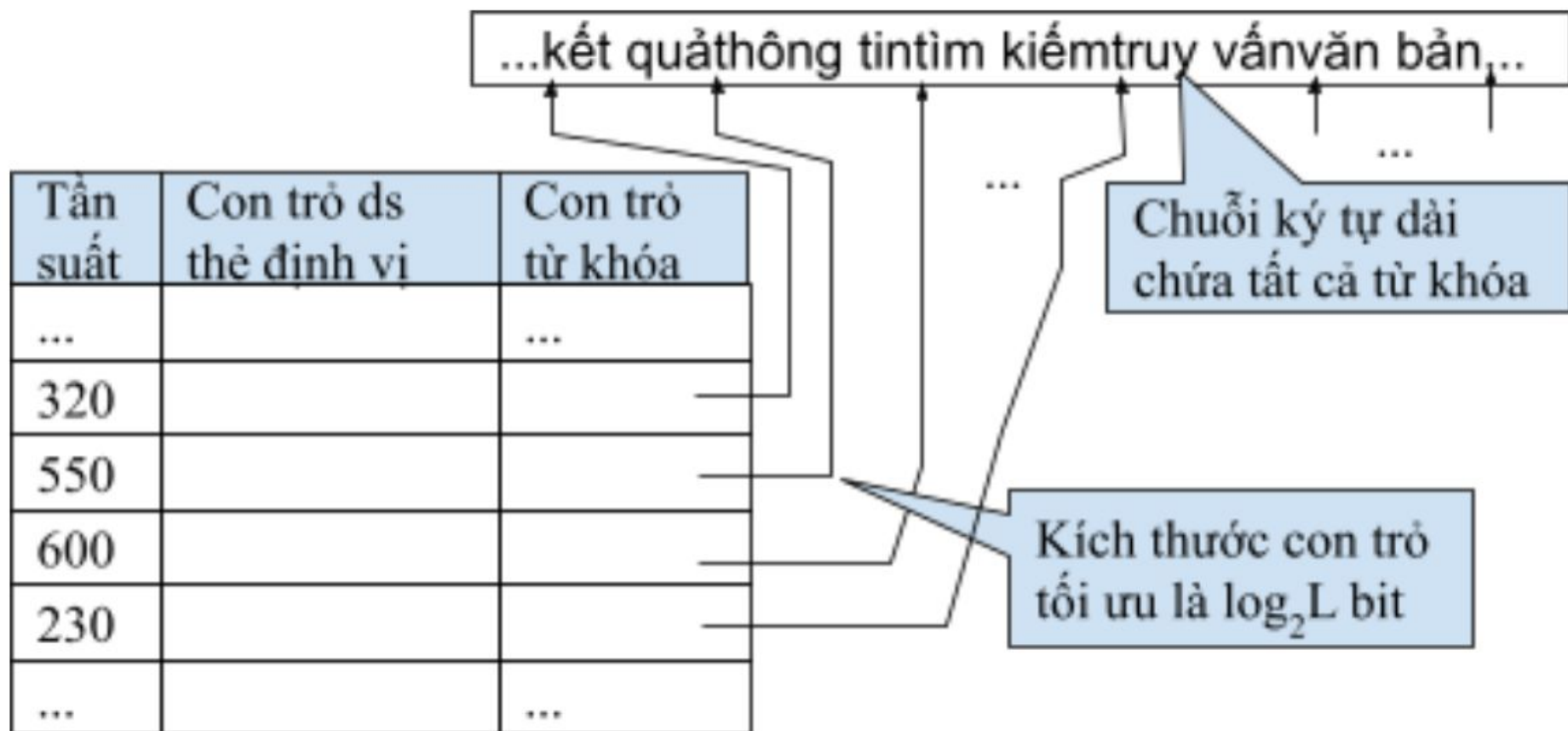
Sử dụng mảng bản ghi có kích thước cố định

- Mỗi mục từ là một bản ghi, từ chỉ mục được biểu diễn như mảng ký tự với độ dài cố định:
  - Ý tưởng đơn giản, có thể truy cập ngẫu nhiên trên ổ đĩa.
  - Tuy nhiên kích thước mảng phải đủ lớn để lưu được từ dài nhất. Có thể có những từ rất dài, tuy nhiên các từ ngắn lại chiếm đa số trong dữ liệu chỉ mục => Có nhiều bytes không dùng đến, lãng phí.



# Biểu diễn chuỗi ký tự dài

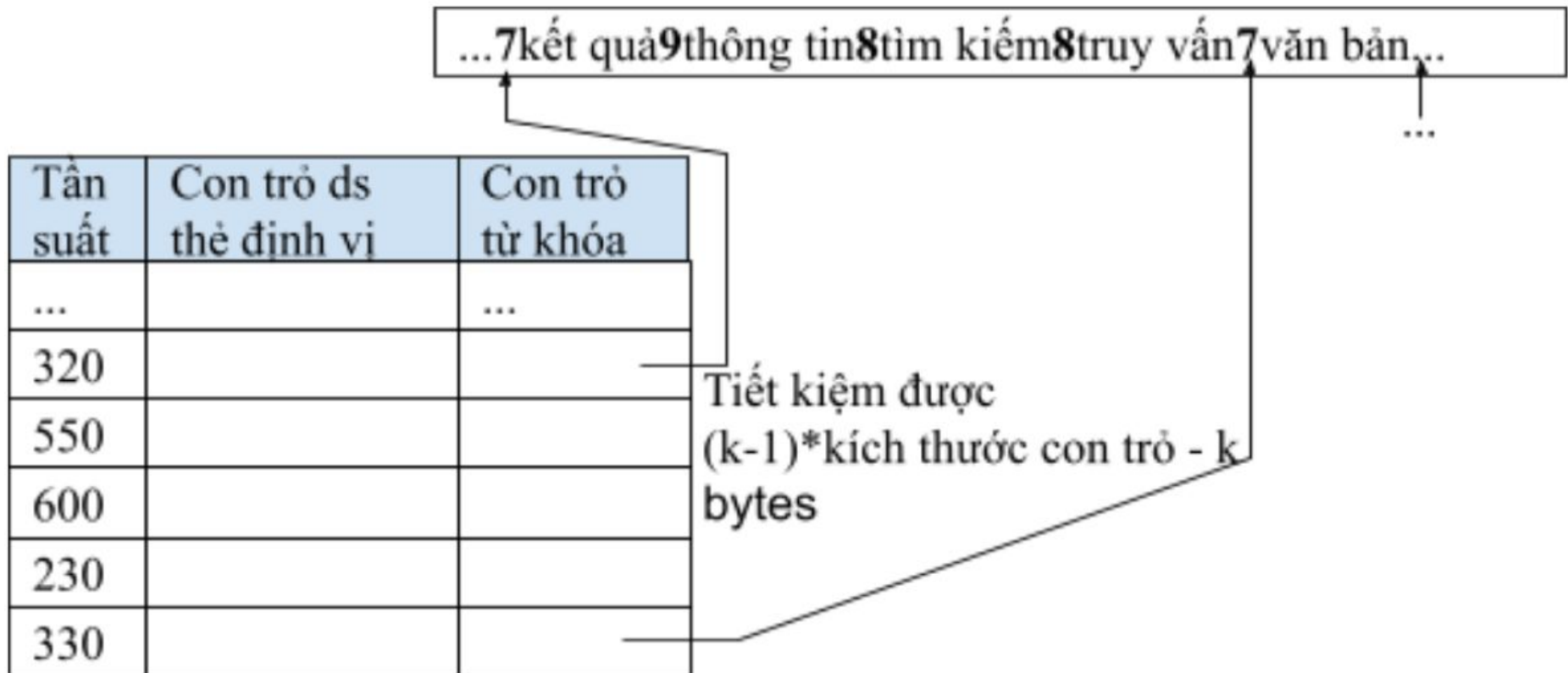
- Lưu bộ từ vựng như một chuỗi ký tự dài :
  - Con trỏ tới từ tiếp theo là dấu hiệu kết thúc từ hiện tại





# Phân đoạn chuỗi ký tự dài

- Chỉ lưu con trỏ tới từ đầu tiên trong khối k từ.
  - Như ví dụ:  $k=4$ .
- Bỏ xung 1 byte để lưu độ dài từ



# Giảm dung lượng do phân đoạn

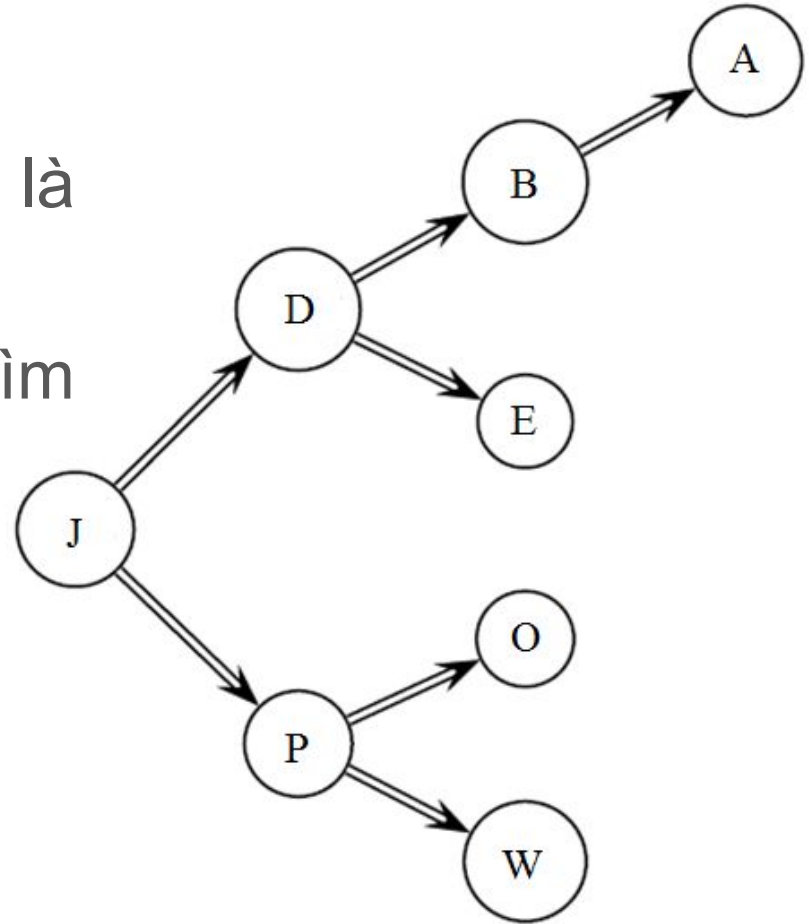
- Giả sử:
  - Kích thước khối  $k = 5$
  - Sử dụng 3 bytes/con trỏ
- Nếu không sử dụng phân đoạn thì cần  $5 * 3 = 15$  bytes để lưu các con trỏ
- Nếu sử dụng phân đoạn thì cần  $3 + 5$  bytes
  - $\Rightarrow$  Tiết kiệm được 7 bytes cho một khối
  - $k$  càng lớn càng tiết kiệm được nhiều bộ nhớ?

*Vì sao không sử dụng  $k$  với giá trị lớn?*

# Tìm kiếm trong từ điển không phân đoạn

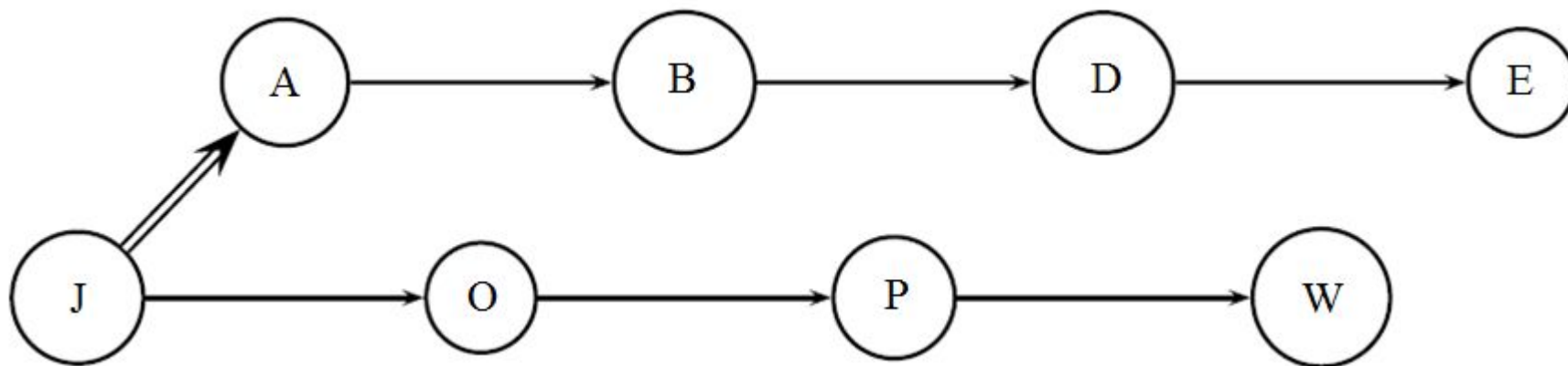
- Giả sử xác suất sử dụng từ là đồng nhất,
- Số so sánh trung bình để tìm một từ là

$$(1+2 \cdot 2+4 \cdot 3+4)/8 \sim 2.6$$



*Nếu phân bố từ truy vấn không đều và chúng ta có các giá trị phân bố đó, thì tổ chức từ điển như thế nào để tìm kiếm từ nhanh nhất?*

# Tìm kiếm trong từ điển có phân đoạn



- Tìm kiếm khối
- Tìm kiếm tuần tự trong mỗi khối.
  - Với khối 4 từ, số so sánh trung bình =  
$$(1+2\cdot 2+2\cdot 3+2\cdot 4+5)/8 = 3 \text{ so sánh}$$

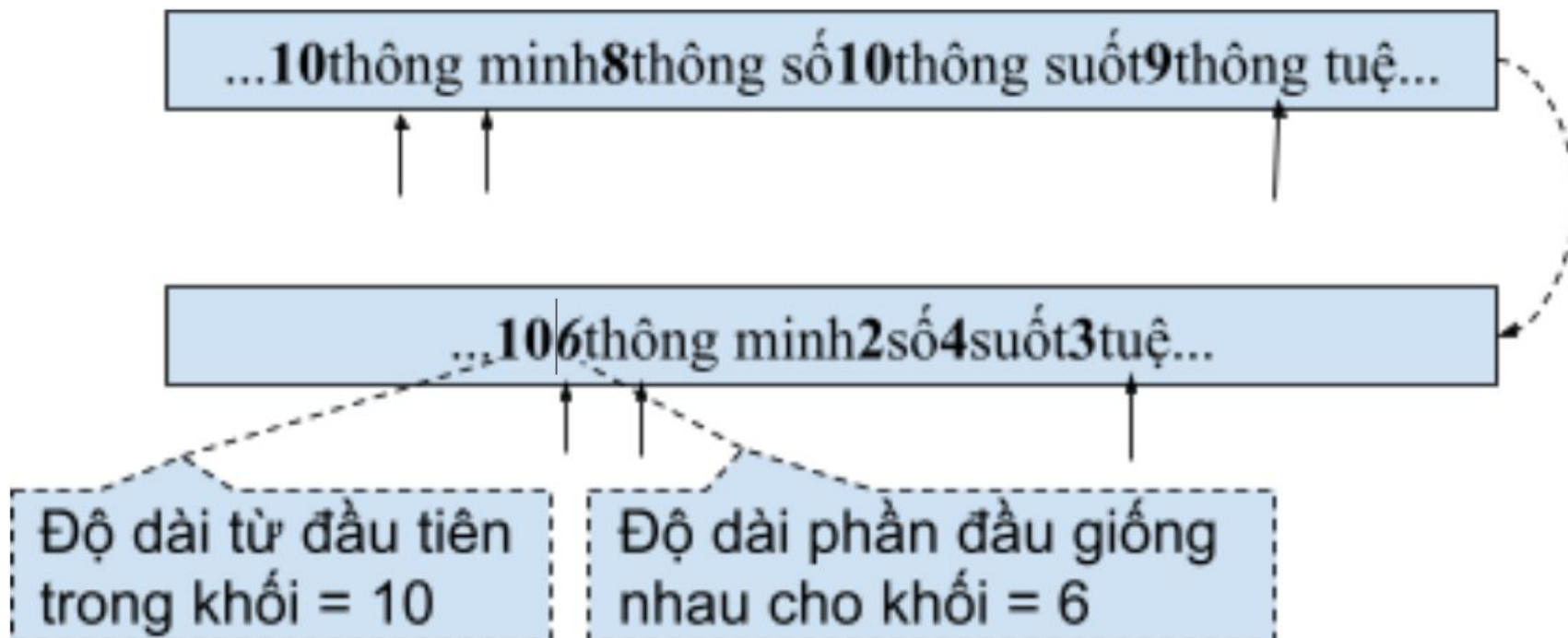
Tốc độ tìm kiếm chậm hơn so với trường hợp không phân đoạn

# Mã hóa đầu từ

- Quan sát:

- Các từ trong một khối thường có phần đầu giống nhau
  - Do các từ được sắp xếp theo thứ tự chữ cái.
  - => Lưu hoàn chỉnh từ đầu tiên trong khối.
  - => Chỉ lưu phần khác biệt cho  $k - 1$  từ còn lại trong khối.

- Ví dụ:



Nén các danh sách thể định vị

# Các quan sát

- Dữ liệu thẻ định vị lớn hơn dữ liệu từ điển rất nhiều (tối thiểu là 10 lần).
  - Nén giúp lưu nhiều thẻ định vị hơn trong bộ nhớ.
- #bits tối thiểu để biểu diễn  $\text{DocID} = (\text{int})\log_2(\text{DocID}) + 1$ 
  - Chúng ta không thể biểu diễn với số lượng bits < số lượng bits tối thiểu (giới hạn nén), tuy nhiên có thể biểu diễn các giá trị nhỏ với số lượng bits ít hơn các giá trị lớn.
- Nếu biểu diễn mã văn bản như số nguyên với độ dài cố định (ví dụ 4 bytes):
  - Với các từ xuất hiện trong ít văn bản (mã văn bản có khác biệt lớn), có thể không tốn nhiều bộ nhớ
  - ... Nhưng với những từ xuất hiện trong hầu hết các văn bản (khác biệt nhỏ giữa các giá trị gần nhau) thì sử dụng vec-tơ đánh dấu có thể hiệu quả hơn

# Danh sách khoảng cách

- Các mã văn bản trong danh sách thẻ định vị được lưu theo trật tự tăng dần, ví dụ:
  - Máy tính: 33, 47, 154, 159, 202 ...
- => có thể thay bằng các khoảng cách với giá trị số nhỏ hơn => sử dụng số lượng bits ít hơn để mã hóa
  - 33, 14, 107, 5, 43 ...
    - Giữ nguyên thẻ định vị đầu tiên trong danh sách
    - Thay các mã văn bản tiếp theo bằng khoảng cách tới mã văn bản ngay trước nó trong danh sách
    - $47 - 33 = 14$
    - $154 - 47 = 107$
    - ...



# Mã hóa với số bytes thay đổi

- Mã hóa với số bytes thay đổi: Variable Bytes Code (Mã VB)
  - Là phương pháp mã hóa hướng tới sử dụng số byte vừa đủ để mã hóa giá trị số (số bytes cần sử dụng để biểu diễn thay đổi theo giá trị số).
- Các ý tưởng cơ bản:
  - Chia 8 bits trong 1 byte thành 2 phần:
    - Sử dụng bit trái nhất (chúng ta gọi là bit c), làm dấu hiệu xác định byte cuối cùng trong dãy bytes biểu diễn giá trị số.
    - Sử dụng 7 bits còn lại để lưu biểu diễn nhị phân của giá trị số.
  - Thiết lập  $c = 1$  cho byte cuối cùng, và  $c = 0$  cho các bytes còn lại.
- Mã VB có giải thuật mã hóa và giải mã tương đối đơn giản với tỉ lệ nén tốt cho số nguyên không âm
  - Được sử dụng trong nhiều hệ thống tìm kiếm;
  - Tương thích tốt với các biểu diễn dữ liệu trong máy tính.

# Mã VB: Mã hóa

```
VBEncodeNumber(n)
```

```
1  bytes = <>
2  while n > 0
3      Prepend(bytes, n mod 128)
6      n = n div 128
7  bytes[length(bytes) - 1] += 128
8  return bytes
```

Xử lý tương đương trên dãy bits thuần nhị phân của số  $n$ :

- Nếu độ dài dãy bits không chia hết cho 7 thì thêm bit 0 vào bên trái cho tới khi độ dài chia hết cho 7.
- Chia dãy bits thành các nhóm 7 bits liên tục.
- Thêm bit  $c$  vào đầu (bên trái) mỗi nhóm ( $c = 0$  cho các nhóm đầu tiên,  $c = 1$  cho nhóm cuối cùng bên phải) để tạo thành 1 byte hoàn chỉnh.
- Kết quả thu được chính là mã VB của số  $n$ .

# Mã VB: Giải mã

- Được thực hiện theo trình tự ngược so với mã hóa
- Dãy bytes được giải mã theo tiến trình tuần tự
  - Đọc tới đâu giải mã tới đó
  - Byte cuối cùng trong biểu diễn của một số có giá trị  $\geq 128$

```
VBDecode(bytestream)
1  numbers = <>
2  n = 0
3  for b in bytestream
4      if b < 128
5          n = n * 128 + b
6      else
7          n = n * 128 + (b - 128)
8          Append(numbers, n)
9          n = 0
10 return numbers
```

# Ví dụ Mã VB

docIDs	824	829	215406
Khoảng cách		5	214577
Mã VB	00000110 10111000	10000101	00001101 00001100 10110001

Dãy số được lưu như một dãy byte liên tiếp

000001101011100010000101000011010000110010110001

Có thể giải mã VB theo tiến trình đọc từ ổ đĩa: Sử dụng đồng thời cả Bus hệ thống và CPU

Các giá trị số nhỏ (5),  
*Sử dụng nguyên một byte nên  
vẫn lãng phí một số bits?*

# Các đơn vị dữ liệu cho mã VB

Có thể sử dụng các đơn vị dữ liệu khác phù hợp với các giá trị số:

- Nếu sử dụng đơn vị dữ liệu lớn (ví dụ word = 2 bytes) thì có thể lãng phí bộ nhớ với các giá trị số nhỏ, và ngược lại nếu sử dụng đơn vị dữ liệu nhỏ (ví dụ nybble = 4 bits) thì có thể lãng phí bộ nhớ với các giá trị số lớn (quá nhiều bit đánh dấu 0).
- Ngoài đơn vị dữ liệu tiêu chuẩn là byte còn có các đơn vị dữ liệu khác: 32 bits, 16 bits, 4 bits v.v..

Ngoài ra cũng có những triển khai gom nhóm nhiều giá trị nhỏ thành một word (2 bytes)

*Mã VB và UTF-8?*

# Mã Unary

- Biểu diễn một số nguyên  $n$  không âm như một dãy  $n$  bits 1 và một bit 0 ở cuối;
- Mã Unary cho số 3 là 1110
- Mã Unary cho số 10 là 1111111110
- Mã Unary cho số 80 là

111

1110

- Trường hợp đặc biệt, mã Unary cho số 0 là 0

Có vẻ không có tiềm năng về tính tối ưu...

... Nhưng dễ dàng xác định bit kết thúc biểu diễn.

*Mã Unary là một thành phần trong mã Gamma (mã  $\gamma$ )*

# Mã Gamma ( $\gamma$ )

- Chúng ta có thể nén tốt hơn với mã hóa ở mức bit
  - Mã  $\gamma$  là một mã phổ biến với xử lý ở mức bit
- Mã  $\gamma$  của một giá trị khoảng cách  $G$  gồm hai thành phần là length và offset
  - offset là dãy số nhị phân còn lại sau khi xóa bit đầu tiên trong biểu diễn thuần nhị phân của  $G$  (luôn bằng 1 với số nguyên dương).
  - Ví dụ  $G = 13 = \underline{1}101_2 \Rightarrow \text{offset} = 101$ ;
  - length là mã Unary của số lượng bits của offset;
  - Ví dụ với offset =  $101_2$  (3 bits)  $\Rightarrow \text{length} = 1110$
  - Mã  $\gamma$  không biểu diễn được số 0.
- Sau khi nối length và offset chúng ta thu được mã  $\gamma$ 
  - Ví dụ mã  $\gamma$  của 13 là 1110101

# Ví dụ mã Gamma ( $\gamma$ )

Giá trị số	length	offset	mã- $\gamma$
0	-	-	-
1	0		0
2	10	0	100
3	10	1	101
4	110	00	11000
9	1110	001	1110001
13	1110	101	1110101
24	11110	1000	111101000
511	111111110	11111111	11111111011111111
1025	11111111110	0000000001	11111111110000000001



# Các tính chất của mã Gamma

- Giá trị khoảng cách  $G$  được mã hoá bằng  
 $2 * \lfloor \log_2 G \rfloor + 1$  bits  
offset chiếm  $\lfloor \log_2 G \rfloor$  bits  
length chiếm  $\lfloor \log_2 G \rfloor + 1$  bits  
=> Gần gấp 2 lần số lượng bits trong biểu diễn tối ưu.
- Số lượng bits trong mã Gamma luôn là số lẻ

# So sánh mã VB và mã Gamma

- Cả 2 đều có thể được giải mã tuần tự theo tiến trình đọc dữ liệu.
- Mã  $\gamma$  có tỉ lệ nén ổn định cho tất cả các phân bố giá trị mã văn bản và nén tốt hơn mã VB;
- Mã  $\gamma$  không có tham số.
- Các kích thước thuận tiện cho xử lý bằng máy tính là 8, 16, 32, 64 bits
  - Các thao tác trên các đơn vị khác sẽ chậm hơn
  - Các xử lý ở mức bits có tốc độ chậm
- Các xử lý cho mã VB sử dụng các đơn vị tiêu chuẩn, vì vậy có thể nhanh hơn mã  $\gamma$ .
- Mã  $\gamma$  ít được sử dụng để nén chỉ mục vì tốc độ xử lý chậm

## Bài tập 3.1. Cấu trúc chỉ mục động

Cho  $n = 2$ , và  $1 \leq T \leq 30$ , hãy phân tích hoạt động của giải thuật LogarithmMerge và vẽ bảng đánh dấu các chỉ mục có trên ổ đĩa (trong số  $I_0, \dots, I_4$ ) ở các thời điểm sau khi xử lý  $2 * k$  từ ( $1 \leq k \leq 15$ ).

Gợi ý: Phần đầu của bảng có các giá trị như sau:

	$I_3$	$I_2$	$I_1$	$I_0$
2	0	0	0	1
4	0	0	1	0

## Bài tập 3.2. Chỉ mục chính&phụ

Đối với giải pháp chỉ mục chính&phụ, chỉ mục phụ có thể ảnh hưởng đáng kể đến tính chính xác của các đại lượng thống kê trên bộ dữ liệu nói chung, và cụ thể là đối với idf, được tính theo công thức  $idf_i = \log(N/df_i)$ .

Hãy lấy một ví dụ chứng minh chỉ mục phụ dù nhỏ cũng có thể gây sai lệch lớn đến idf nếu chỉ tính toán trong phạm vi chỉ mục chính.

*Gợi ý:* xét một từ hiếm nhưng đột ngột xuất hiện thường xuyên.

## Bài tập 3.3. Mã hóa danh sách MVB

Cho danh sách mã văn bản sau:

1, 3, 10, 120, 121

- a) Hãy xác định danh sách khoảng cách;
- b) Hãy mã hóa kết quả mục a bằng mã VB, đơn vị mã hóa là 8 bits;
- c) Hãy mã hóa kết quả mục a bằng mã Gamma.

*Kết quả là dãy bits*

## Bài tập 3.4. Giải mã danh sách MVB

a) Dãy bits sau là mã VB của **danh sách khoảng cách**, đơn vị mã hóa là 4 bits (nibbles):

1010 0001 1011 0101 1000

Hãy xác định **danh sách mã văn bản ban đầu**.

b) Dãy bits sau là mã Gamma của **danh sách khoảng cách**:

110011110000101

Hãy xác định **danh sách mã văn bản ban đầu**.

## Bài tập 3.5. Một số luật thống kê từ vựng

Hãy chứng minh nếu luật Zipf đúng thì kích thước bộ từ vựng là hữu hạn,

và nếu luật Heap đúng thì kích thước bộ từ vựng là vô hạn.

Chúng ta có thể suy diễn luật Heap từ luật Zipf hay không?

## Bài tập 3.6. Ước lượng kích thước từ điển

Cho một bộ từ vựng với 100 000 từ. Từ dài nhất chiếm 20 bytes, tần suất văn bản chiếm 4 bytes, và con trỏ đến danh sách thẻ định vị chiếm 4 bytes. Trung bình mỗi từ chiếm 8 bytes.

Hãy ước lượng tỉ lệ nén từ điển cho hai trường hợp: Chia khối với  $k = 8$  và  $k = 16$ .



## Bài tập 3.7. Nén từ điển

Cho khối từ chỉ mục: Lập trình C, Lập trình C++, Lập trình Java, Lập trình Python.

- a) Hãy viết biểu diễn nén sử dụng phân đoạn và mã hóa đầu từ cho khối từ chỉ mục đã cho.
- b) Tính tỉ lệ dung lượng dữ liệu cho kết quả mục a so với tổng dung lượng các từ ban đầu.

## Bài tập 3.8. Giải nén từ điển

Cho khối từ chỉ mục đã được mã hóa đầu từ như sau:

7|**10**Tiếng Anh4Nga5Nhật5Pháp

Hãy giải mã để lấy khối từ ban đầu.

*Ghi chú:* Chúng ta giả sử một ký tự Unicode như ê, â (mã hóa UTF-8) chiếm 2 bytes.

