Introdução à Recuperação de Informações

https://github.com/fccoelho/curso-IRI

IRI 3: Dicionários e Recuperação Tolerante

Flávio Codeço Coelho

Escola de Matemática Aplicada, Fundação Getúlio Vargas

Sumário

Distinguindo entre Tipo e token

- Token – Uma instância de uma palavra ou termo ocorrendo em um documento
- Tipo – Uma classe de equivalência de tokens
- *In June, the dog likes to chase the cat in the barn.*
- 12 tokens, 9 tipos de palavras

Problemas na tokenização

- Quais são os delimitadores? Espaços? Apóstrofes? Hífen?
- Para cada um destes: às vezes eles delimitam, às vezes não.
- Muitas línguas não possuem espaços! (P.ex., Chinês)
- Não Há espaços em palavras compostas em Holandês, Alemão e Sueco (*Lebensversicherungsgesellschaftsangestellter*)

Problemas com classes de equivalência

- Um termo é uma classe de equivalência de tokens.
- Como definir Classes de equivalência?
- Números: (3/20/91 vs. 20/3/91)
- Capitalização
- Truncagem, Truncador de Porter
- Análise Morfológica : inflecional vs. derivacional
- Problemas de classes de equivalências em outras línguas
  - Morfologias mais complexas do que o inglês
  - Finlandês: Um único verbo pode ter 12000 formas diferentes
  - Acentos, tremas, etc.

Principais conclusões de hoje

Principais conclusões de hoje

- Recuperação Tolerante: O que fazer se não há correspondência exata entre o termo de consulta e os termos do documento.

Principais conclusões de hoje

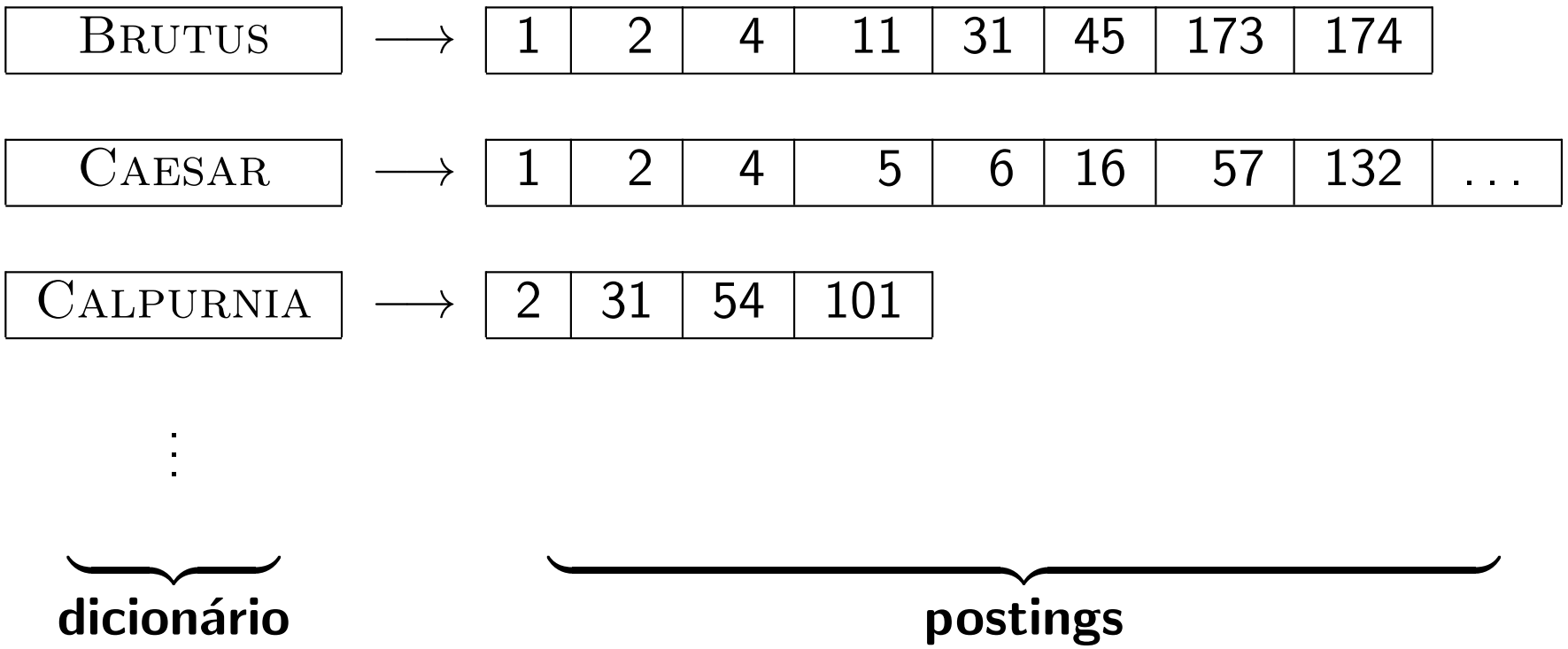
- Recuperação Tolerante: O que fazer se não há correspondência exata entre o termo de consulta e os termos do documento.
- Consultas coringas

Principais conclusões de hoje

- Recuperação Tolerante: O que fazer se não há correspondência exata entre o termo de consulta e os termos do documento.
- Consultas coringas
- Correção ortográficas

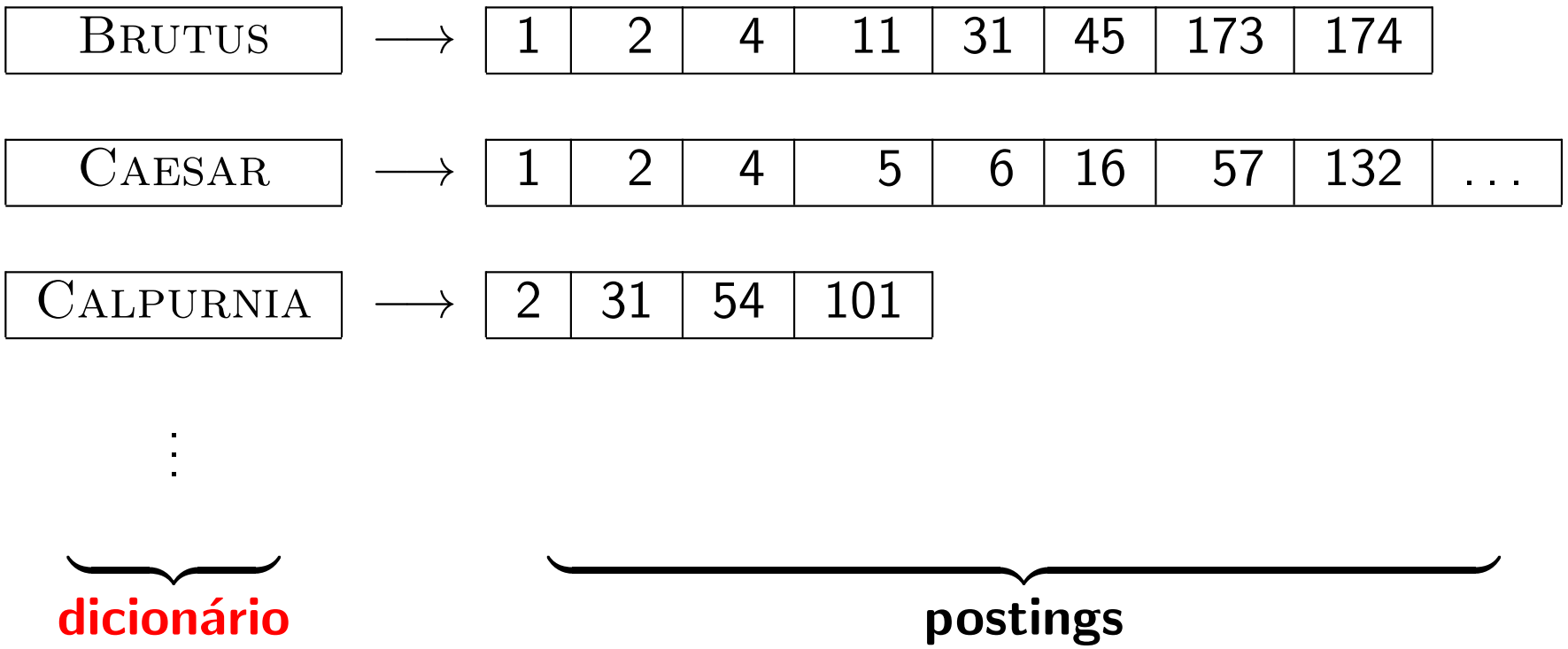
Índice invertido

Para cada termo t, armazenamos uma lista de documentos que contém t.



Índice invertido

Para cada termo t, armazenamos uma lista de documentos que contém t.



Dicionários

- O dicionário é a estrutura de dados que é usada para armazenar o vocabulário de termos.

Dicionários

- O dicionário é a estrutura de dados que é usada para armazenar o vocabulário de termos.
- vocabulário de termos: os dados

Dicionários

- O dicionário é a estrutura de dados que é usada para armazenar o vocabulário de termos.
- vocabulário de termos: os dados
- Dicionário: A estrutura de dados para armazenamento do vocabulário

Dicionário como uma matriz com entradas de tamanho fixo

- Para cada termo, precisamos armazenar um par de ítens:

Dicionário como uma matriz com entradas de tamanho fixo

- Para cada termo, precisamos armazenar um par de ítens:
  - Frequência de documentos

Dicionário como uma matriz com entradas de tamanho fixo

- Para cada termo, precisamos armazenar um par de ítens:
  - Frequência de documentos
  - Ponteiro para a lista de postings

Dicionário como uma matriz com entradas de tamanho fixo

- Para cada termo, precisamos armazenar um par de ítens:
  - Frequência de documentos
  - Ponteiro para a lista de postings
  - ...

Dicionário como uma matriz com entradas de tamanho fixo

- Para cada termo, precisamos armazenar um par de ítens:
  - Frequência de documentos
  - Ponteiro para a lista de postings
  - ...
- Assuma por ora que podemos armazenar esta informação em uma entrada de tamanho fixo.

Dicionário como uma matriz com entradas de tamanho fixo

- Para cada termo, precisamos armazenar um par de ítens:
  - Frequência de documentos
  - Ponteiro para a lista de postings
  - ...
- Assuma por ora que podemos armazenar esta informação em uma entrada de tamanho fixo.
- Assuma que armazenamos estas entradas em uma matriz.

Dicionário como uma matriz com entradas de tamanho fixo

termo	documento frequência	ponteiro para lista de postings
a	656.265	$\longrightarrow$
aachen	65	$\longrightarrow$
...	...	...
zulu	221	$\longrightarrow$
espaço necessário: 20 bytes	4 bytes	4 bytes

como acessamos um termo de consulta q_i nesta matriz em tempo de consulta? Ou seja: Que estrutura de dados usamos para localizar a entrada (linha) na matriz onde q_i está armazenado?

Estruturas de dados para acessar os termos

- Duas Classes principais de estruturas de dados: hashes e árvores

Estruturas de dados para acessar os termos

- Duas Classes principais de estruturas de dados: hashes e árvores
- Alguns sistemas de RI usam hashes, outros usam árvores.

Estruturas de dados para acessar os termos

- Duas Classes principais de estruturas de dados: hashes e árvores
- Alguns sistemas de RI usam hashes, outros usam árvores.
- Critérios de escolha:

Estruturas de dados para acessar os termos

- Duas Classes principais de estruturas de dados: hashes e árvores
- Alguns sistemas de RI usam hashes, outros usam árvores.
- Critérios de escolha:
  - Existe um número fixo de termos ou ele crescerá indefinidamente?

Estruturas de dados para acessar os termos

- Duas Classes principais de estruturas de dados: hashes e árvores
- Alguns sistemas de RI usam hashes, outros usam árvores.
- Critérios de escolha:
  - Existe um número fixo de termos ou ele crescerá indefinidamente?
  - Quais as frequências relativas com que as várias chaves serão acessadas?

Estruturas de dados para acessar os termos

- Duas Classes principais de estruturas de dados: hashes e árvores
- Alguns sistemas de RI usam hashes, outros usam árvores.
- Critérios de escolha:
  - Existe um número fixo de termos ou ele crescerá indefinidamente?
  - Quais as frequências relativas com que as várias chaves serão acessadas?
  - Quantos termos teremos?

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante
- Prós: Busca em um hash é mais rápida do que em uma árvore.

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante
- Prós: Busca em um hash é mais rápida do que em uma árvore.
  - Tempo de consulta é constante.

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante
- Prós: Busca em um hash é mais rápida do que em uma árvore.
  - Tempo de consulta é constante.
- Contras

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante
- Prós: Busca em um hash é mais rápida do que em uma árvore.
  - Tempo de consulta é constante.
- Contras
  - Não há forma de encontrar pequenas variações (*resume* vs. *résumé*)

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante
- Prós: Busca em um hash é mais rápida do que em uma árvore.
  - Tempo de consulta é constante.
- Contras
  - Não há forma de encontrar pequenas variações (*resume* vs. *résumé*)
  - Não permite busca de prefixos (Todos os termos que começam com *automat*)

Hashes

- Cada termo do vocabulário é "hasheado" para um inteiro.
- Busca-se evitar colisões
- No momento da consulta, faz-se o seguinte: Hasheia o termo de consulta, resolve as colisões, Localiza entrada em uma matriz de elementos com tamanho constante
- Prós: Busca em um hash é mais rápida do que em uma árvore.
  - Tempo de consulta é constante.
- Contras
  - Não há forma de encontrar pequenas variações (*resume* vs. *résumé*)
  - Não permite busca de prefixos (Todos os termos que começam com *automat*)
  - é necessário "rehashear" tudo periodicamente se o vocabulário continua crescendo.

Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).

Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).
- Árvore mais simples: árvore binária

Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).
- Árvore mais simples: árvore binária
- Busca é ligeiramente mais lenta que em hashes: $O(\log M)$, onde M é o tamanho do vocabulário.

Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).
- Árvore mais simples: árvore binária
- Busca é ligeiramente mais lenta que em hashes: $O(\log M)$, onde M é o tamanho do vocabulário.
- $O(\log M)$ vale apenas para árvores balanceadas.

Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).
- Árvore mais simples: árvore binária
- Busca é ligeiramente mais lenta que em hashes: $O(\log M)$, onde M é o tamanho do vocabulário.
- $O(\log M)$ vale apenas para árvores balanceadas.
- Rebalancear árvores binárias é caro.

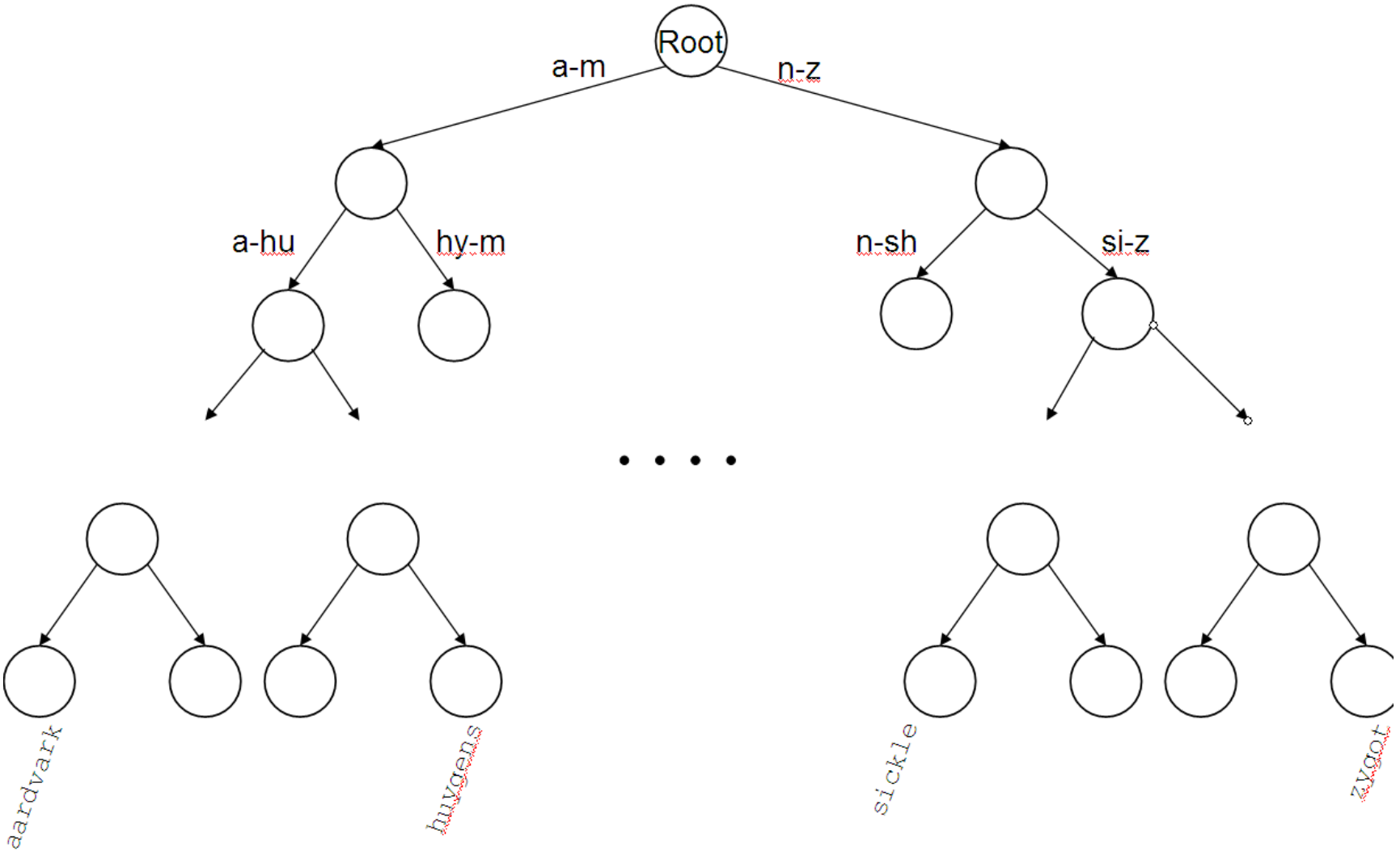
Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).
- Árvore mais simples: árvore binária
- Busca é ligeiramente mais lenta que em hashes: $O(\log M)$, onde M é o tamanho do vocabulário.
- $O(\log M)$ vale apenas para árvores balanceadas.
- Rebalancear árvores binárias é caro.
- Arvores-B resolvem o problema do balanceamento.

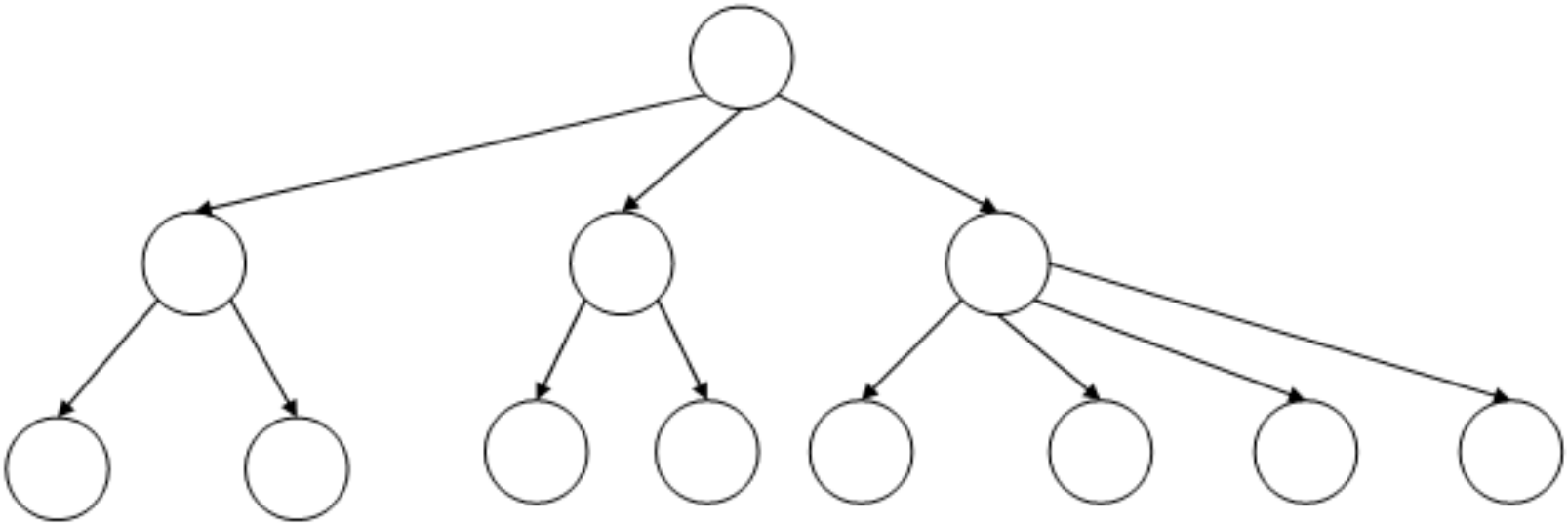
Árvores

- Árvores resolvem o problema do prefixo (Encontrar todos os termos começando com *automat*).
- Árvore mais simples: árvore binária
- Busca é ligeiramente mais lenta que em hashes: $O(\log M)$, onde M é o tamanho do vocabulário.
- $O(\log M)$ vale apenas para árvores balanceadas.
- Rebalancear árvores binárias é caro.
- Arvores-B resolvem o problema do balanceamento.
- Definição de árvore-B: cada nó interno tem um número de filhos no intervalo $[a, b]$ onde a, b são inteiros positivos apropriados , p.ex., $[2, 4]$.

Árvore Binária



Árvore B



Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*

Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*
- Fácil com dicionários baseados em árvore B: recupera todos os termos t no intervalo: $\text{mon} \leq t < \text{moo}$

Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*
- Fácil com dicionários baseados em árvore B: recupera todos os termos t no intervalo: mon $\leq t <$ moo
- *mon: Encontre todos os documentos contendo termos que terminam com *mon*

Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*
- Fácil com dicionários baseados em árvore B: recupera todos os termos t no intervalo: mon $\leq t <$ moo
- *mon: Encontre todos os documentos contendo termos que terminam com *mon*
  - Mantém uma árvore adicional para termos *ao contrário*

Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*
- Fácil com dicionários baseados em árvore B: recupera todos os termos t no intervalo: mon $\leq t <$ moo
- *mon: Encontre todos os documentos contendo termos que terminam com *mon*
  - Mantém uma árvore adicional para termos *ao contrário*
  - Então recupera todos os termos t no intervalo: nom $\leq t <$ non

Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*
- Fácil com dicionários baseados em árvore B: recupera todos os termos t no intervalo: mon $\leq t <$ moo
- *mon: Encontre todos os documentos contendo termos que terminam com *mon*
  - Mantém uma árvore adicional para termos *ao contrário*
  - Então recupera todos os termos t no intervalo: nom $\leq t <$ non
- Resultado: Um conjunto de termos que correspondem à consulta coringa

Consultas coringa

- mon*: Encontre todos os documentos contendo termos começados por *mon*
- Fácil com dicionários baseados em árvore B: recupera todos os termos t no intervalo: mon $\leq t <$ moo
- *mon: Encontre todos os documentos contendo termos que terminam com *mon*
  - Mantém uma árvore adicional para termos *ao contrário*
  - Então recupera todos os termos t no intervalo: nom $\leq t <$ non
- Resultado: Um conjunto de termos que correspondem à consulta coringa
- Então recupera todos os documentos que contenham estes termos

Como lidar com um * no meio de um termo

- Exemplo: m*nchen

Como lidar com um * no meio de um termo

- Exemplo: m*nchen
- Poderíamos buscar todos os termos que satisfazem m* e *nchen Na árvore B e reter a interseção dos dois conjuntos.

Como lidar com um * no meio de um termo

- Exemplo: m*nchen
- Poderíamos buscar todos os termos que satisfazem m* e *nchen Na árvore B e reter a interseção dos dois conjuntos.
- Mas sai caro

Como lidar com um * no meio de um termo

- Exemplo: m*nchen
- Poderíamos buscar todos os termos que satisfazem m* e *nchen Na árvore B e reter a interseção dos dois conjuntos.
- Mas sai caro
- Alternativa: índice permuterm

Como lidar com um * no meio de um termo

- Exemplo: m*nchen
- Poderíamos buscar todos os termos que satisfazem m* e *nchen Na árvore B e reter a interseção dos dois conjuntos.
- Mas sai caro
- Alternativa: índice permuterm
- Idéia básica: Rotaciona cada consulta coringa, de forma que o * ocorra no final.

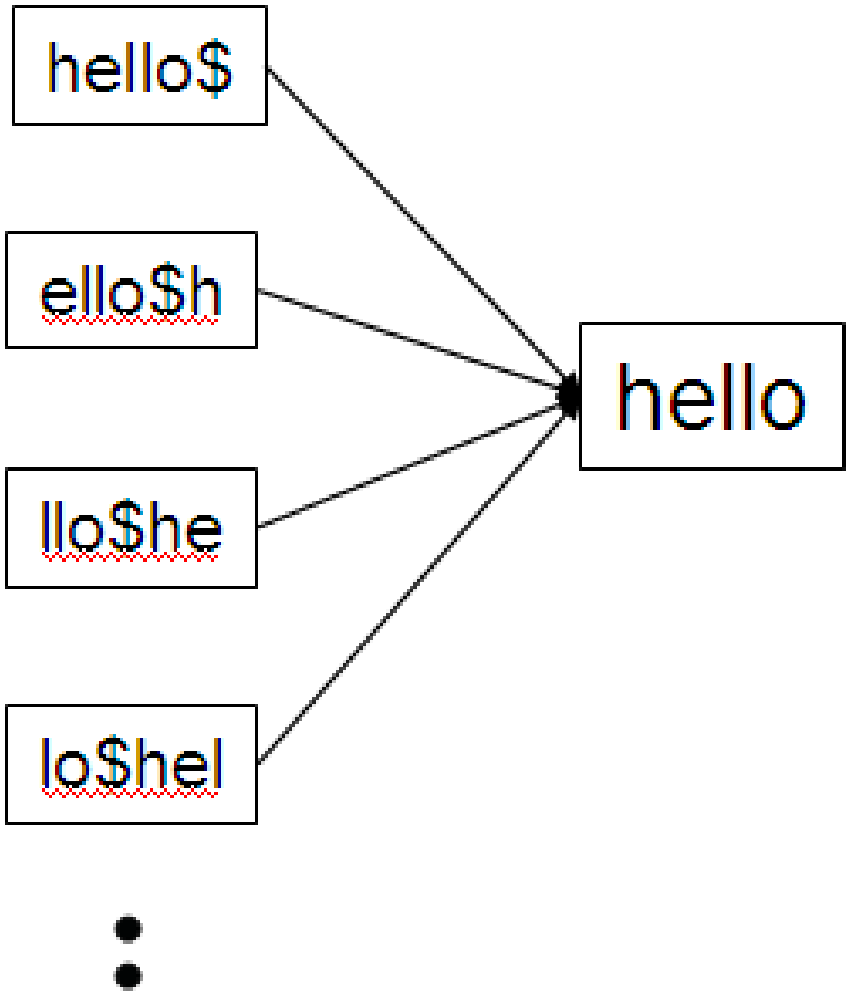
Como lidar com um * no meio de um termo

- Exemplo: m*nchen
- Poderíamos buscar todos os termos que satisfazem m* e *nchen Na árvore B e reter a interseção dos dois conjuntos.
- Mas sai caro
- Alternativa: índice permuterm
- Idéia básica: Rotaciona cada consulta coringa, de forma que o * ocorra no final.
- Armazena cada uma destas rotações no dicionário, por exemplo, em uma árvore B

Índice Permuterm

- Para o termo HELLO: adicione *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, e *$hello* à árvore B onde $ é um símbolo especial

Permuterm → mapeamento de termos



Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$*

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$*
  - For *X*, look up X*

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$*
  - For *X*, look up X*
  - For X*Y, look up Y$X*

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$*
  - For *X*, look up X*
  - For X*Y, look up Y$X*
  - Example: For hel*o, look up o$hel*

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$*
  - For *X*, look up X*
  - For X*Y, look up Y$X*
  - Example: For hel*o, look up o$hel*
- Permuterm index would better be called a permuterm tree.

Permuterm index

- For HELLO, we've stored: *hello$*, *ello$h*, *llo$he*, *lo$hel*, and *o$hell*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$*
  - For *X*, look up X*
  - For X*Y, look up Y$X*
  - Example: For hel*o, look up o$hel*
- Permuterm index would better be called a permuterm tree.
- But permuterm index is the more common name.

Processing a lookup in the permuterm index

- Rotate query wildcard to the right

Processing a lookup in the permuterm index

- Rotate query wildcard to the right
- Use B-tree lookup as before

Processing a lookup in the permuterm index

- Rotate query wildcard to the right
- Use B-tree lookup as before
- Problem: Permuterm more than quadruples the size of the dictionary compared to a regular B-tree. (empirical number)