Information Retrieval
& Data Mining

Information Retrieval & Data Mining
Universität des Saarlandes, Saarbrücken
Wintersemester 2013/14

Chapter III:
Ranking Principles

Zipf's Law (after George Kingsley Zipf)

- The **collection frequency** cf_i of the i-th most frequent word in the document collection is **inversely proportional** to the rank i

$$cf_i \propto \frac{1}{i}$$

- For the relative collection frequency with **language-specific constant** c (for English $c \approx 0.1$) we obtain

$$\frac{cf_i}{\sum_j cf_j} \propto \frac{c}{i}$$

- In an English document collection, we can thus expect the most frequent word to account for 10% of all term occurrences

George Kingsley Zipf

Levenshtein Edit Distance

- **Levenshtein edit distance** between two strings *x* and *y* is the minimal number of edit operations (*insert*, *replace*, *delete*) required to transform *x* into *y*

- The minimal number of operations $m[i, j]$ to transform the **prefix substring** $x[1{:}i]$ into $y[1{:}j]$ is defined via the **recurrence**

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j] \; ? \; 0 \; : \; 1) & \text{(replace } x[i]\text{?)} \\ m[i-1, j] \; + \; 1 & \text{(delete } x[i]\text{)} \\ m[i, j-1] \; + \; 1 & \text{(insert } y[j]\text{)} \end{cases}$$

  and can be computed using **dynamic programming**

- <u>Examples</u>: *d*(*house, rose*) = 2

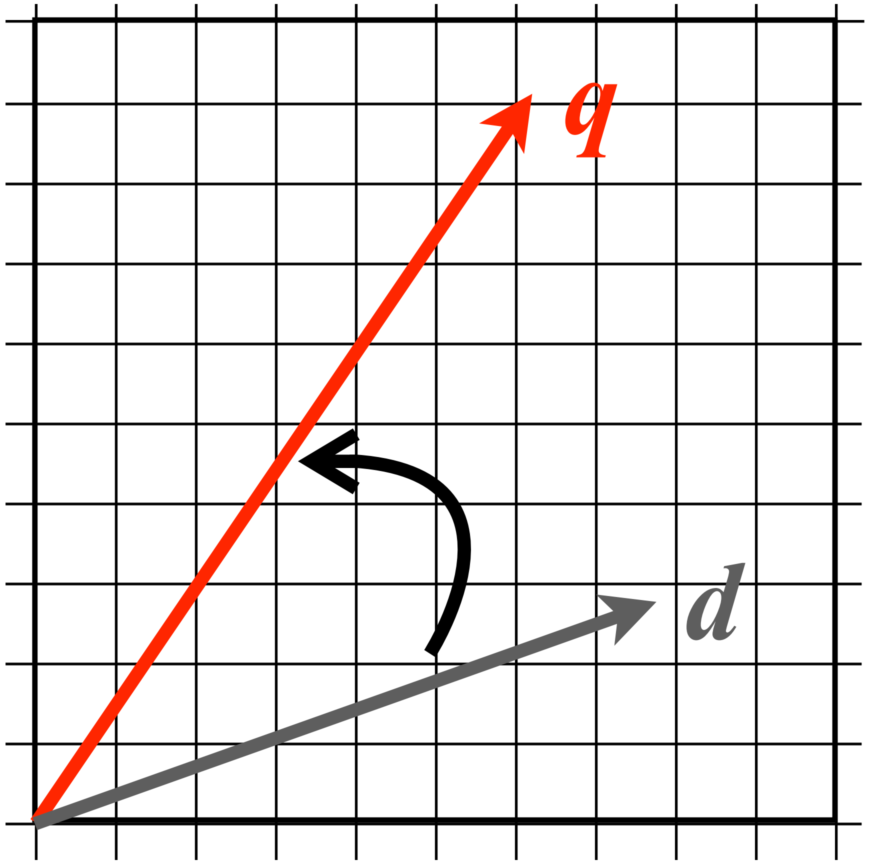
Vector Space Model (VSM)

- Boolean retrieval model provides **no (or only rudimentary) ranking of results** – severe limitation for large result sets
- Vector space model views **documents and queries as vectors** in a $|V|$-dimensional vector space (i.e., one dimension per term)
- **Cosine similarity** between two vectors ***q*** and ***d*** is the cosine of the angle between them

$$sim(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \, \|\mathbf{d}\|}$$

$$= \frac{\sum_{i=1}^{|V|} \mathbf{q}_i \, \mathbf{d}_i}{\sqrt{\sum_{i=1}^{|V|} \mathbf{q}_i^2} \sqrt{\sum_{i=1}^{|V|} \mathbf{d}_i^2}}$$

$$= \frac{\mathbf{q}}{\|\mathbf{q}\|} \frac{\mathbf{d}}{\|\mathbf{d}\|}$$



TF*IDF

- **Term frequency** $tf_{t,d}$ as
  the number of times the term t occurs in document d

- **Document frequency** df_t as
  the number of documents that contain the term t

- **Inverse document frequency** idf_t as

$$idf_t = \frac{|D|}{df_t}$$

  with $|D|$ as the **number of documents** in the collection

- The tf.idf weight of term t in document d is then defined as

$$tf.idf_{t,d} = tf_{t,d} \times idf_t$$

  favoring terms that **occur often in the document** d
  and/or **not in many documents from the collection** D

Precision, Recall, and Accuracy

- **Precision** *P* is the fraction of retrieved documents that is relevant

$$P = \frac{tp}{tp + fp}$$

- **Recall** *R* is the fraction of relevant results that is retrieved

$$R = \frac{tp}{tp + fn}$$

- **Accuracy** *A* is the fraction of correctly classified documents

$$A = \frac{tp + tn}{tp + fp + tn + fn}$$

Not appropriate for IR

(Mean) Average Precision

- Precision, recall, and F-measure ignore the order of results
- **Average precision** (AP) averages over retrieved relevant results
  - Let $\{d_1, \ldots, d_{mj}\}$ be the set of relevant results for the query q_j
  - Let R_{jk} be the set of ranked retrieval results for the query q_j from top until you get to the relevant result d_k

$$\mathrm{AP}(q_j) = \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk})$$

- **Mean average precision** (MAP) averages over multiple queries

$$\mathrm{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \mathrm{AP}(q_j)$$

(Normalized) Discounted Cumulative Gain

- What if we have **graded labels** as relevance assessments? (e.g., 0 : not relevant, 1 : marginally relevant, 2 : relevant)

- **Discounted cumulative gain** (DCG) for query *q*

$$DCG(q,k) = \sum_{m=1}^{k} \frac{2^{R(q,m)} - 1}{log(1+m)}$$

  with $R(q, m) \in \{0, \ldots, 2\}$ as label of m-th retrieved result

- **Normalized discounted cumulative gain** (NDCG)

$$NDCG(q,k) = \frac{DCG(q,k)}{IDCG(q,k)}$$

  normalized by **idealized discounted cumulative gain** (IDCG)

(Normalized) Discounted Cumulative Gain

- *IDCG*(*q, k*) is the **best-possible** value *DCG*(*q, k*) achievable for the query *q* on the document collection at hand

- Example: Let *R*(*q, m*) $\in$ {0, …, 2} and assume that two documents have been labeled with 2, two with 1, all others with 0. The best-possible top-5 result thus has labels < 2, 2, 1, 1, 0 > and determines the value of *IDCG*(*q, k*) for this query

- NDCG **also considers rank** at which relevant results are retrieved

- NDCG is typically averaged over **multiple queries**

$$NDCG(Q,k) = \frac{1}{|Q|} \sum_{q \in Q} NDCG(q,k)$$

Okapi BM25

- **State-of-the-art retrieval model** (among top-ranked in TREC) having roots in **Probabilistic Information Retrieval**

$$w_{t,d} = \frac{(k_1 + tf_{t,d})}{k_1((1-b) + b\,\frac{|d|}{avdl}) + tf_{t,d}} \; log \, \frac{|D| - df_j + 0.5}{df_j + 0.5}$$

  - k_1 controls **impact of term frequency** (common choice $k_1 = 1.2$)
  - b controls **impact of document length** (common choice $b = 0.75$)

Multinomial Language Model

- Query q is seen as a **bag of terms** and generated from document d by **drawing terms** from the bag of terms corresponding to d

$$
\begin{aligned}
P(q|d) &= \binom{|q|}{tf(t_1, q) \ldots tf(t_{|q|}, q)} \prod_{t_i \in q} P(t_i|d)^{\,tf(t_i,q)} \\
&\propto \prod_{t_i \in q} P(t_i|d)^{\,tf(t_i,q)} \\
&\approx \prod_{t_i \in q} P(t_i|d) \quad (\textbf{assuming } \forall t_i \in q \, : \, tf(t_i, q) = 1)
\end{aligned}
$$

- **Maximum-likelihood estimate** for parameters $P(t_i|d)$

$$
P(t_i|d) = \frac{tf(t_i, d)}{|d|}
$$

Smoothing

- **Jelinek-Mercer smoothing** as **linear combination** of document language model θ_d and document-collection language model θ_D

$$P(t|d) = \lambda \, \frac{tf(t,d)}{|d|} + (1-\lambda) \, \frac{tf(t,D)}{|D|}$$

with document D as concatenation of entire document collection

- **Dirichlet-prior smoothing** with a **conjugate Dirichlet prior** instead of the Maximum-Likelihood Estimation

$$P(t|d) = \frac{tf(t,d) + \alpha \, \frac{tf(t,D)}{|D|}}{|d| + \alpha}$$

Chapter IV: Link Analysis

PageRank

- **Random surfer model**
  - follows a uniform random outgoing link with probability (1-ε)
  - jumps to a uniform random web page with probability ε
- Matrix **T** captures following of a uniform random outgoing link

$$\mathbf{T}_{ij} = \begin{cases} 1/out(i) & : \quad (i,j) \in E \\ 0 & : \quad \text{otherwise} \end{cases}$$

- Vector **j** captures jumping to a uniform random web page

$$\mathbf{j}_i = 1/|V|$$

- **Transition probability matrix** of Markov chain then obtained as

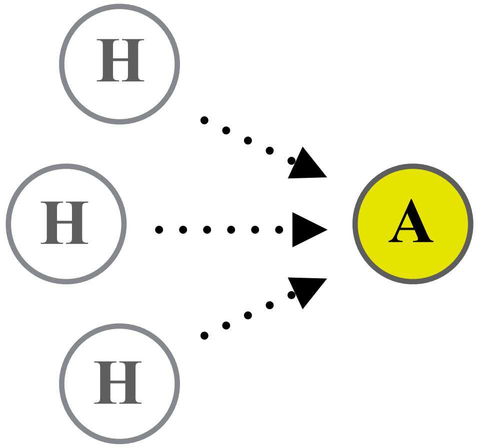
$$\mathbf{P} = (1-\epsilon)\,\mathbf{T} + \epsilon \begin{bmatrix} 1 & \ldots & 1 \end{bmatrix}^T \mathbf{j}$$

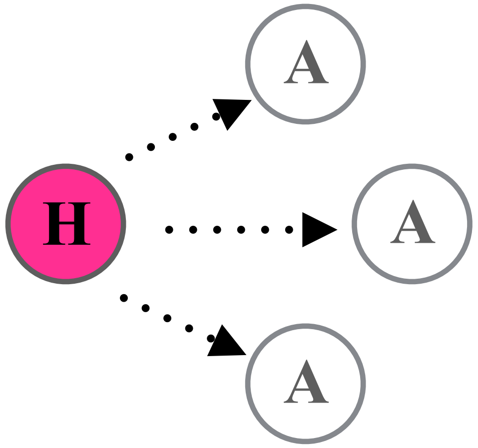
HITS

- **Hyperlinked-Induced Topic Search** (HITS) identifies
  - **authorities** as good content sources (~high indegree)
  - **hubs** as good link sources (~high outdegree)
- **HITS** [Kleinberg '99] considers a web page
  - a **good authority** if many **good hubs link to it**
  - a **good hub** if it **links to many good authorities**

  ~ **mutual reinforcement** between hubs & authorities

Jon Kleinberg





HITS

- Given (partial) Web graph $G(V, E)$, let $a(v)$ and $h(v)$ denote the **authority score** and **hub score** of the web page v

$$a(v) \propto \sum_{(u,v) \in E} h(u) \qquad\qquad h(v) \propto \sum_{(v,w) \in E} a(w)$$

- Authority and hub scores in **matrix notation**

$$\boldsymbol{a} = \alpha \, \boldsymbol{A}^T \boldsymbol{h} \qquad\qquad \boldsymbol{h} = \beta \, \boldsymbol{A} \, \boldsymbol{a}$$

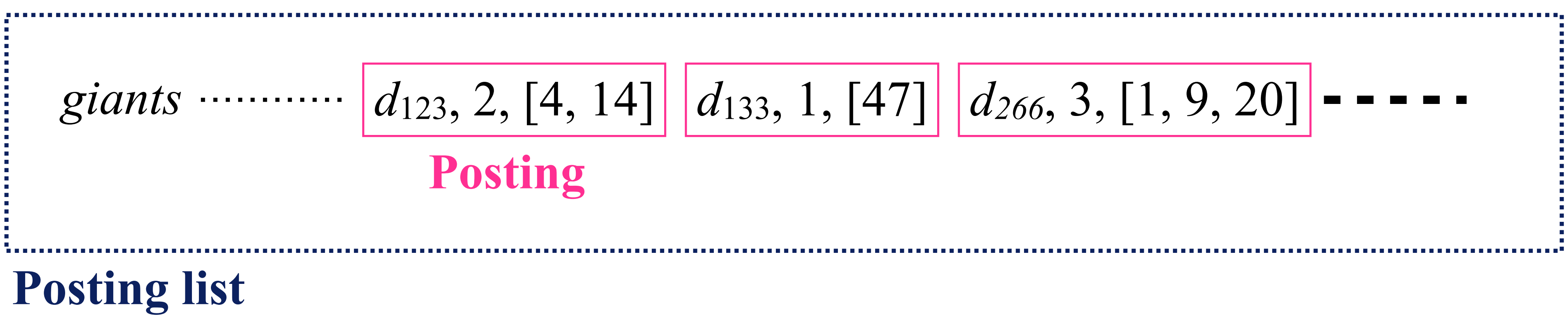
  with adjacency matrix $\boldsymbol{A}$, hub vector $\boldsymbol{a}$, authority vector $\boldsymbol{h}$, and constants α and β

- Authority vector $\boldsymbol{a}$ and hub vector $\boldsymbol{h}$ are **eigenvectors** of **cocitation matrix** $\boldsymbol{A}^T\boldsymbol{A}$ and **coreference matrix** $\boldsymbol{A}\boldsymbol{A}^T$

Chapter V:
Indexing & Searching

Inverted Index

- Inverted index keeps a **posting list** for each term, which usually reside on secondary storage, with each **posting** capturing information about term's **occurrences in a specific document**
  - **document identifier** (e.g., d_{123}, d_{234}, …)
  - **term frequency** (e.g., $tf(house, d_{123}) = 2$, $tf(house, d_{234}) = 4$)
  - **score impacts** (e.g., $tf(house, d_{123}) * idf(house) = 3.75$)
  - **offsets** (i.e., absolute positions at which the term occurs in the document)



- Posting lists are usually **compressed** for time and space efficiency

Inverted Index

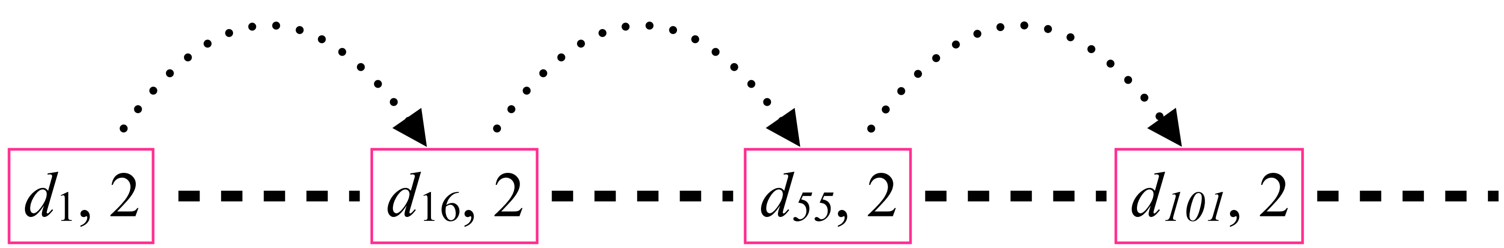
- **Document-ordered posting lists** for more efficient intersections (e.g., required for Boolean queries and phrase queries)

d_{123}, 2, [4, 14] | d_{133}, 1, [47] | d_{266}, 3, [1, 9, 20] - - - - -

- **Impact-ordered posting lists** for more efficient top-*k* queries (i.e., terminate query processing as soon as top-*k* result is known)

d_{231}, 1.0 | d_{12}, 0.9 | d_{662}, 0.8 | d_{3}, 0.5 - - - - -

- **Skip pointers** allow "fast forwarding" in a posting list



Ziv-Lempel Compression

- **LZ77** (Adaptive Dictionary) and further variants:
  - Scan text and identify in a **lookahead window** the longest string that occurs repeatedly and is contained in **backwards window**
  - Replace this string by a **pointer** to its previous occurrence
- Encode text into list of **triples < back, count, new >** where
  - **back** is the backward distance to a prior occurrence of the string that starts at the current position
  - **count** is the length of this repeated string
  - **new** is the next symbol that follows the repeated string
- Triples themselves can be further encoded (with variable length)
- Variants use explicit dictionary with statistical analysis of text but need to scan text twice (for statistics and compression)

Variable-Byte Encoding

- 32-bit binary code represents 12,038 using 4 bytes as

00000000	00000000	00**101111**	00000**11**0

- **Variable-byte encoding** (aka. 7-bit encoding) uses one bit per byte as a continuation bit indicating whether the current number expands into the next bytes

- Variable-byte encoding represents 12,038 using only 2 bytes as

0**1**0**1111**0	**1**00000**11**0

1 continuation bit

7 data bits

- **Byte-aligned**, i.e., each number corresponds to sequence of bytes

Gamma Encoding

- Gamma (γ) encoding represents an integer x as
  - *length* = *floor*($\log_2 x$) in **unary**
  - *offset* = x - 2^{length} in **binary**

  results in ($1 + \log_2 x + \log_2 x$) bits for integer x

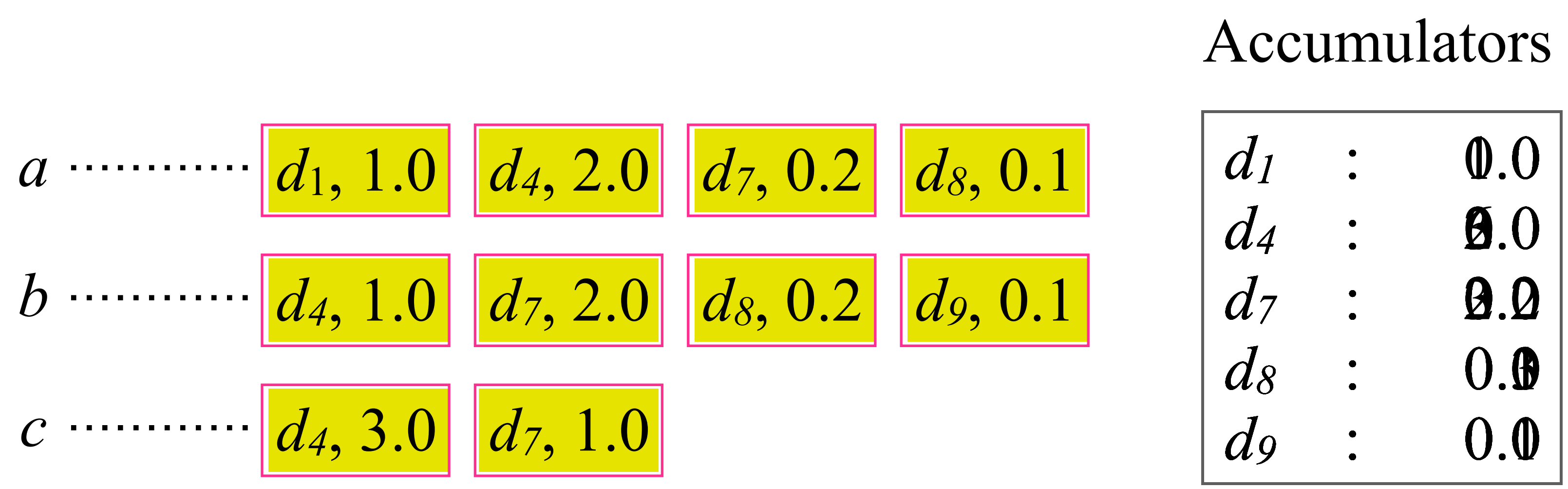
- **Not byte-aligned**, i.e., needs to be packed into bytes or words

- Useful when **distribution** of numbers is **not known** ahead of time or when **small numbers** (e.g., gaps, tf) are **frequent**

Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **successively**
  - maintains an **accumulator** for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d) \text{ after the first } j \text{ posting lists have been read}$$



  - required **memory** depends on the **number of accumulators** maintained
  - **top-*k* results** can be determined by **sorting accumulators** at the end

Document-at-a-Time Query Processing

- **Document-at-a-Time** (DAAT) query processing
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **concurrently**
  - computes score when **same document** is seen in one or more posting lists

a	d_1, 1.0	d_4, 2.0	d_7, 0.2	d_8, 0.1
b	d_4, 1.0	d_7, 2.0	d_8, 0.2	d_9, 0.1
c	d_4, 3.0	d_7, 1.0		

d_1	:	1.0
d_4	:	6.0
d_7	:	3.2
d_8	:	0.3
d_9	:	0.1

  - always advances posting list with **lowest current document identifier**
  - required main memory depends on the **number of results** to be reported
  - **top-*k* results** can be determined by keeping results in **priority queue**

Fagin's Threshold Algorithms

- **Threshold Algorithm** (TA)
  - original version, often used as synonym for entire family of algorithms
  - requires eager random access to candidate objects
  - worst-case memory consumption: $O(k)$
- **No-Random-Accesses** (NRA)
  - no random access required, may have to scan large parts of the lists
  - worst-case memory consumption: $O(m*n + k)$

Fagin's Threshold Algorithms

- Assume **score-ordered posting lists** and **additional index** for score look-ups by document identifier
- Scan posting lists using **inexpensive sequential accesses** (SA) in round-robin manner
- Perform **expensive random accesses** (RA) to look up scores for a specific document when beneficial
- Support **monotone score aggregation function**

$$aggr : \mathbb{R}^m \rightarrow \mathbb{R} \; : \; \forall x_i \geq x_i' \Rightarrow aggr(x_1, \ldots, x_m) \geq aggr(x_1', \ldots, x_m')$$

- Compute **aggregate scores** incrementally in **candidate queue**
- Compute **score bounds** for candidate results and stop when **threshold test** guarantees correct top-*k* result

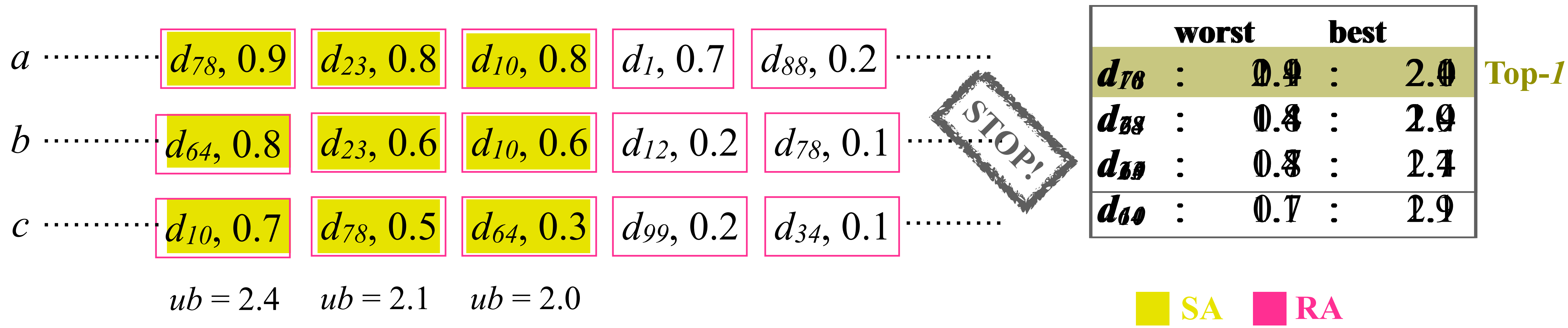
No-Random-Accesses Algorithm (NRA)

- **Sequential accesses** (SA) only
- Worst-case memory consumption $O(m*n + k)$

<u>No-Random-Accesses Algorithm</u> (NRA):
scan index lists (e.g., round-robin)
consider $d = cdid(i)$ in posting list for t_i
$high(i) = cscore(i)$
$eval(d) = eval(d) \cup \{i\}$ // where have we seen d?

$worst(d) = aggr\{\ score(t_j, d) \mid j \in eval(d)\ \}$
$best(d) = aggr\{\ worst(d), aggr\{\ high(j) \mid j \notin eval(d)\ \}\ \}$

if $worst(d) > min_k$ **then** // good enough for top-k?
  add d top top-k
  $min_k = min\{\ worst(d') \mid d' \in$ top-k $\}$
else if $best(d) > min_k$ **then** // good enough for cand?
  $cand = cand \cup \{\ d\ \}$
$ub = max\{\ best(d') \mid d' \in cand\ \}$
if $ub \leq min_k$ **then**
  exit

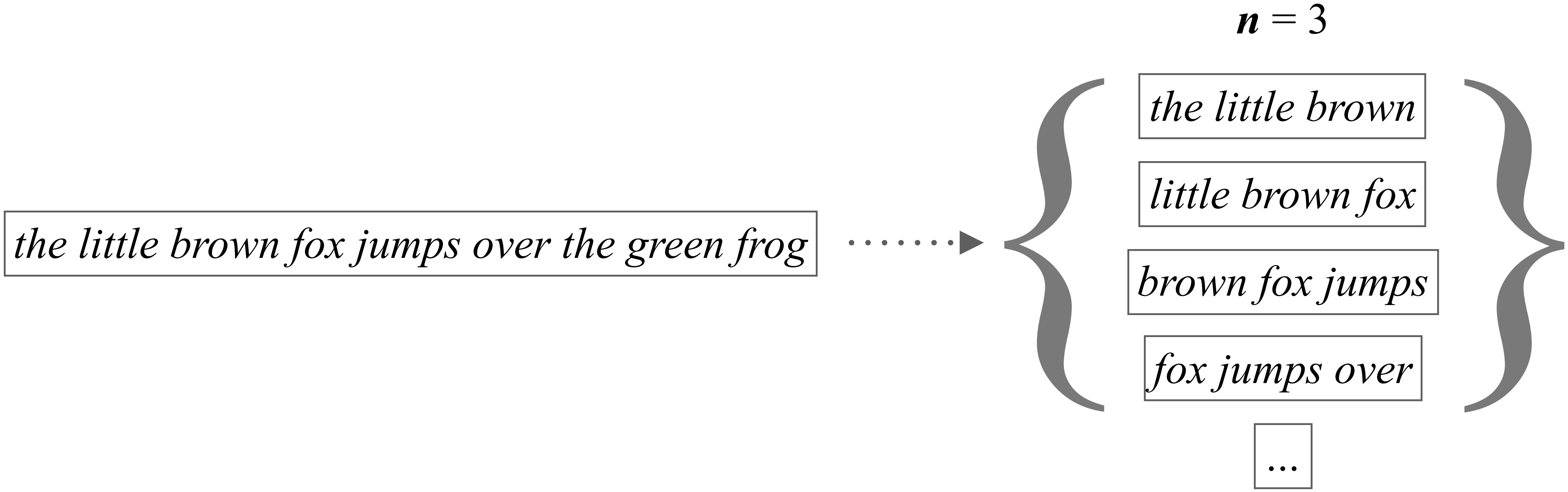


Shingling

- Observation: Duplicates on the Web are often **slightly perturbed** (e.g., due to different boilerplate, minor rewordings, etc.)

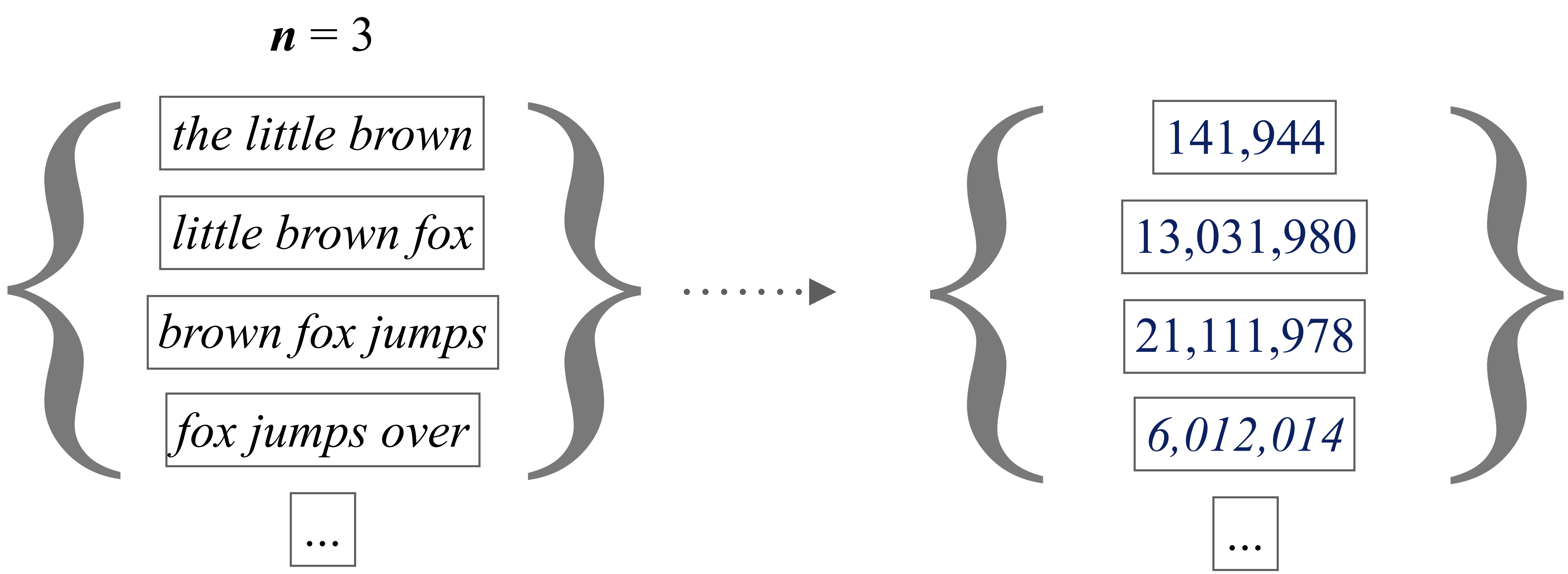
- **Document fingerprinting** (e.g., SHA-1 or MD5) is not effective, since we need to allow for minor differences between documents

- **Shingling** represents document d as set $S(d)$ of **word-level *n*-grams** (*shingles*) and compares documents based on these sets



Shingling

- Encode shingles by **hash fingerprints** (e.g., using SHA-1), yielding a set of numbers $S(d) \subseteq [1, \ldots, n]$ (e.g., for $n = 2^{64}$)



- Compare suspected near-duplicate documents d and d' by
  - **Resemblance** $\frac{|S(d) \cap S(d')|}{|S(d) \cup S(d')|}$ (Jaccard coefficient)
  - **Containment** $\frac{|S(d) \cap S(d')|}{|S(d)|}$ (Relative overlap)

Min-Wise Independent Permutations

- **Statistical sketch** to estimate the resemblance of $S(d)$ and $S(d')$
  - consider m **independent random permutations** of the two sets, implemented by applying m **independent hash functions**
  - keep the **minimum value** observed for each of the m hash functions, yielding a m-dimensional MIPs vector for each document
  - **estimate resemblance** of $S(d)$ and $S(d')$ based on MIPs(d) and MIPs(d')

$$\hat{r}(d, d') = \frac{|\{1 \leq i \leq m \mid MIPs(d)[i] = MIPs(d')[i]\}|}{m}$$

- Full details: [Broder et al. '00]

Min-Wise Independent Permutations

Set of shingle fingerprints

$S(d) = \{ 3, 8, 12, 17, 21, 24\}$

$\boldsymbol{h_1(x) = 7x + 3 \bmod 51}$
$\{ 24, 8, 36, 20, 48, 18\}$

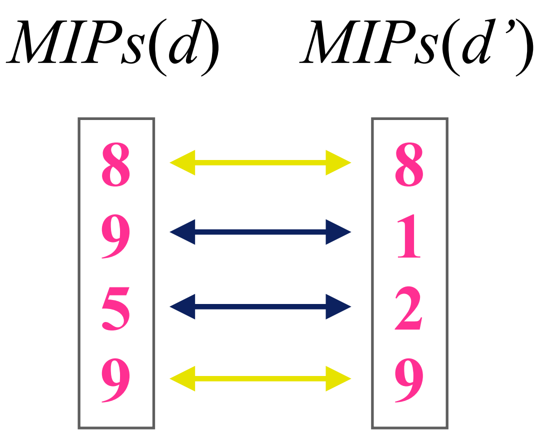
$\boldsymbol{h_2(x) = 5x + 6 \bmod 51}$
$\{ 21, 46, 15, 40, 9, 24\}$

.
.
.

$\boldsymbol{h_m(x) = 3x + 9 \bmod 51}$
$\{ 18, 33, 45, 9, 21, 30\}$

8
9
.
.
.
9

MIPs(*d*)
MIPs vector



Estimated resemblance: 2 / 4

- MIPs are an **unbiased estimator of resemblance**

$$P[min\{h(x)|x \in A\} = min\{h(y)|y \in B\}] = |A \cap B|/|A \cup B|$$

- MIPs can be seen as **repeated random sampling** of x,y from A,B

Chapter VI:
Information Extraction

Hidden Markov Models (HMMs)

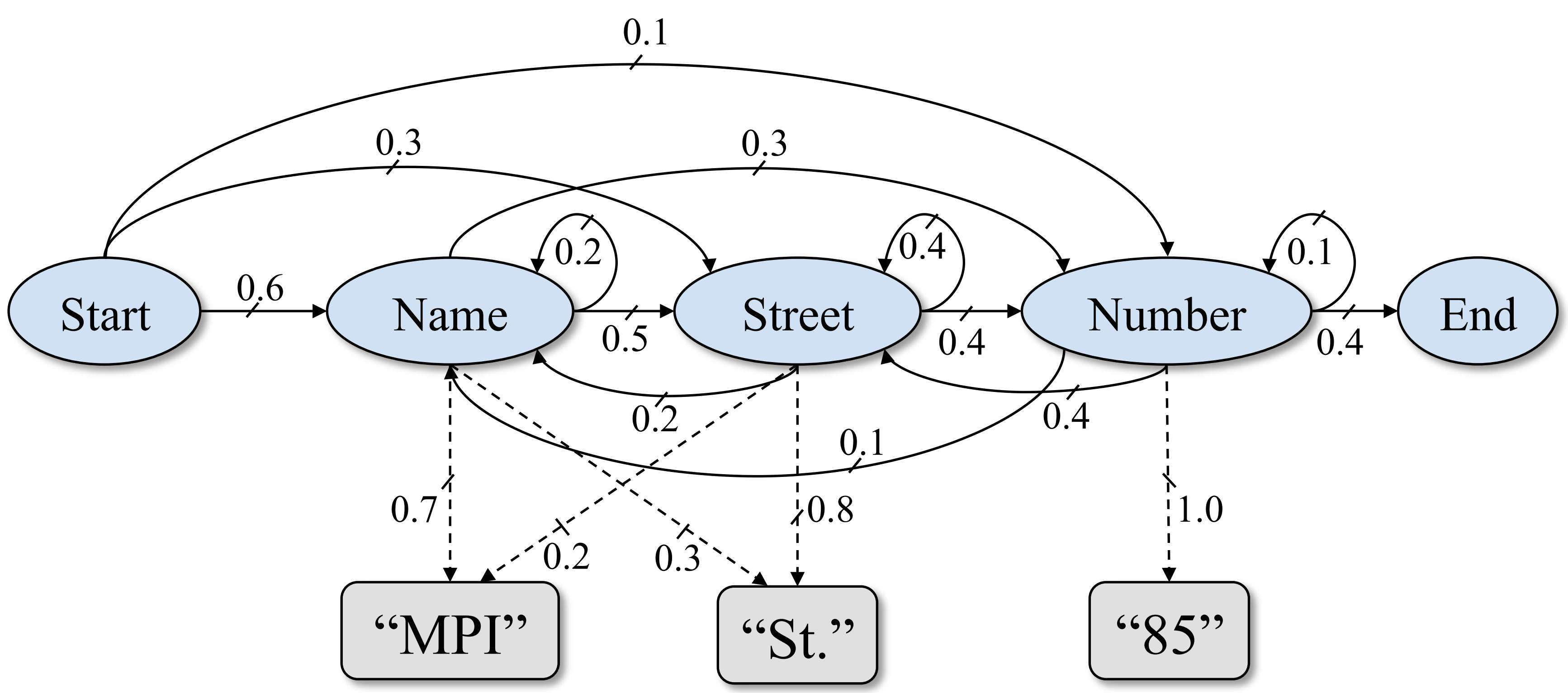
- Hidden Markov Model (HMM) is a discrete-time, finite-state Markov model consisting of
  - **state space** $S = \{s_1, \ldots, s_n\}$ and the state in step t is denoted as $X(t)$
  - **initial state probabilities** p_i ($i = 1, \ldots, n$)
  - **transition probabilities** $p_{ij} : S \times S \rightarrow [0,1]$, denoted $p(s_i \rightarrow s_j)$
  - **output alphabet** $\Sigma = \{w_1, \ldots, w_m\}$
  - **state-specific output probabilities** $q_{ik} : \mathrm{S} \times \Sigma \rightarrow [0,1]$, denoted $q(\mathrm{s}_i \uparrow w_k)$
- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$

$$\sum_{x_1,\ldots,x_T \in S} \prod_{i=1}^{T} p(x_{i-1} \rightarrow x_i)\, q(x_i \uparrow o_i) \text{ with } p(x_0 \rightarrow x_i) = p(x_i)$$

HMM Example

- <u>Goal</u>: Label the tokens in the sequence
  Max-Planck-Institute, Stuhlsatzenhausweg 85
  with the labels **Name**, **Street**, **Number**

$\mathbf{\Sigma}$ = {"MPI", "St.", "85"} // output alphabet
S = {Name, Street, Number} // (hidden) states
$\boldsymbol{p_i}$ = {0.6, 0.3, 0.1} // initial state probabilities



Forward Computation

- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$ is

$$\sum_{x_1,\ldots,x_T \in S} \prod_{i=1}^{T} p(x_{i-1} \rightarrow x_i)\, q(x_i \uparrow o_i) \text{ with } p(x_0 \rightarrow x_i) = p(x_i)$$

- **Naïve computation** would require $O(n^T)$ operations!
- **Iterative forward computation** with clever caching and reuse of intermediate results ("memoization") requires $O(n^2\, T)$ operations
  - Let $\alpha_i(t) = P[o_1, \ldots, o_{t-1}, X(t) = i]$ denote the probability of being in state i at time t and having already emitted the prefix output $o_1, \ldots, o_{t-1}$
  - Begin: $\alpha_i(1) = p_i$
  - Induction: $\alpha_j(t+1) = \sum_{i=1}^{n} \alpha_i(t)\, p(s_i \rightarrow s_j)\, p(s_i \uparrow o_t)$

Viterbi Algorithm

- Goal: Identify state sequence $x_1, \ldots, x_T$ **most likely of having generated the observed output** $o_1, \ldots, o_T$

- **Viterbi algorithm** (dynamic programming)

$$\delta_i(1) = p_i \qquad \text{// highest probability of being in state } i \text{ at step 1}$$

$$\psi_i(1) = 0 \qquad \text{// highest-probability predecessor of state } i$$

for $t = 1, \ldots, T$

$$\delta_j(t+1) = \max_{i=1,\ldots,n} \delta_i(t)\, p(x_i \rightarrow x_j)\, q(x_i \uparrow o_t) \quad \text{// probability}$$

$$\psi_j(t+1) = \arg\max_{i=1,\ldots,n} \delta_i(t)\, p(x_i \rightarrow x_j)\, q(x_i \uparrow o_t) \quad \text{// state}$$

- Most likely state sequence can be obtained by means of **backtracking** through the memoized values $\delta_i(t)$ and $\psi_i(t)$

Thanks!