PV030 Textual Information Systems

Petr Sojka

Faculty of Informatics
Masaryk University, Brno

Spring 2013

Outline (Week five)

1. Fuzzy (proximity) search. Metrics for measurement of distance of strings.
2. Classification of search: 6D space of search problems.
3. Examples of creation of search engines.
4. Completion of the chapter about searching without text preprocessing.
5. Indexing basics.

Outline (Week five)

1. Fuzzy (proximity) search. Metrics for measurement of distance of strings.
2. Classification of search: 6D space of search problems.
3. Examples of creation of search engines.
4. Completion of the chapter about searching without text preprocessing.
5. Indexing basics.

Outline (Week five)

1. Fuzzy (proximity) search. Metrics for measurement of distance of strings.
2. Classification of search: 6D space of search problems.
3. Examples of creation of search engines.
4. Completion of the chapter about searching without text preprocessing.
5. Indexing basics.

Outline (Week five)

1. Fuzzy (proximity) search. Metrics for measurement of distance of strings.
2. Classification of search: 6D space of search problems.
3. Examples of creation of search engines.
4. Completion of the chapter about searching without text preprocessing.
5. Indexing basics.

Outline (Week five)

1. Fuzzy (proximity) search. Metrics for measurement of distance of strings.
2. Classification of search: 6D space of search problems.
3. Examples of creation of search engines.
4. Completion of the chapter about searching without text preprocessing.
5. Indexing basics.

Part I

Proximity search

Fuzzy search: metrics

Classification of search problems

SFOECO, QFOECO, SSOECO

QSOECO, SFORCO, SFODCO

Metrics (for proximity search)

How to measure (metrics) the similarity of strings?

Definition: we call $d : S \times S \longrightarrow R$ ***metrics*** if the following is true:

1. $d(x, y) \geq 0$
2. $d(x, x) = 0$
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, y) = 0 \Rightarrow x = y$ (identity of indiscernibles)
5. $d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality)

We call the values of the function d (distance).

Metrics (for proximity search)

How to measure (metrics) the similarity of strings?

Definition: we call $d : S \times S \longrightarrow R$ ***metrics*** if the following is true:

1. $d(x, y) \geq 0$
2. $d(x, x) = 0$
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, y) = 0 \Rightarrow x = y$ (identity of indiscernibles)
5. $d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality)

We call the values of the function d (distance).

Metrics for proximity search

Definition: let us have strings X and Y over the alphabet Σ. The minimal number of editing operation for transformation X to Y is

- ***Hamming distance***, R-distance, when we allow just the operation **R**eplace,
- ***Levenshtein distance***, DIR-distance, when we allow the operations **D**elete, **I**nsert and **R**eplace,
- ***Generalized Levenshtein distance***, $DIRT$-distance, when we allow the operations **D**elete, **I**nsert, **R**eplace and **T**ranspose. Transposition is possible at the neighbouring characters only.

They are ***metrics***, Hamming must be performed over strings of the same length, Levenshtein can be done over the different lengths.

Metrics for proximity search

Definition: let us have strings X and Y over the alphabet Σ. The minimal number of editing operation for transformation X to Y is

- ***Hamming distance***, R-distance, when we allow just the operation **R**eplace,
- ***Levenshtein distance***, DIR-distance, when we allow the operations **D**elete, **I**nsert and **R**eplace,
- ***Generalized Levenshtein distance***, $DIRT$-distance, when we allow the operations **D**elete, **I**nsert, **R**eplace and **T**ranspose. Transposition is possible at the neighbouring characters only.

They are ***metrics***, Hamming must be performed over strings of the same length, Levenshtein can be done over the different lengths.

Metrics for proximity search

Definition: let us have strings X and Y over the alphabet Σ. The minimal number of editing operation for transformation X to Y is

- ***Hamming distance***, R-distance, when we allow just the operation **R**eplace,
- ***Levenshtein distance***, DIR-distance, when we allow the operations **D**elete, **I**nsert and **R**eplace,
- ***Generalized Levenshtein distance***, $DIRT$-distance, when we allow the operations **D**elete, **I**nsert, **R**eplace and **T**ranspose. Transposition is possible at the neighbouring characters only.

They are ***metrics***, Hamming must be performed over strings of the same length, Levenshtein can be done over the different lengths.

Metrics for proximity search

Definition: let us have strings X and Y over the alphabet Σ. The minimal number of editing operation for transformation X to Y is

- ***Hamming distance***, R-distance, when we allow just the operation **R**eplace,
- ***Levenshtein distance***, DIR-distance, when we allow the operations **D**elete, **I**nsert and **R**eplace,
- ***Generalized Levenshtein distance***, $DIRT$-distance, when we allow the operations **D**elete, **I**nsert, **R**eplace and **T**ranspose. Transposition is possible at the neighbouring characters only.

They are ***metrics***, Hamming must be performed over strings of the same length, Levenshtein can be done over the different lengths.

Proximity search—examples

Example: Find such an example of strings X and Y, that simultaneously holds $R(X, Y) = 5$, $DIR(X, Y) = 5$, and $DIRT(X, Y) = 5$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y, that holds simultaneously $R(X, Y) = 5$, $DIR(X, Y) = 4$, and $DIRT(X, Y) = 3$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y of the length $2n$, $n \in N$, that $R(X, Y) = 2n$ and a) $DIR(X, Y) = 2$; b) $DIRT(X, Y) = \lceil \frac{n}{2} \rceil$

Proximity search—examples

Example: Find such an example of strings X and Y, that simultaneously holds $R(X, Y) = 5$, $DIR(X, Y) = 5$, and $DIRT(X, Y) = 5$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y, that holds simultaneously $R(X, Y) = 5$, $DIR(X, Y) = 4$, and $DIRT(X, Y) = 3$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y of the length $2n$, $n \in N$, that $R(X, Y) = 2n$ and a) $DIR(X, Y) = 2$; b) $DIRT(X, Y) = \lceil \frac{n}{2} \rceil$

Proximity search—examples

Example: Find such an example of strings X and Y, that simultaneously holds $R(X, Y) = 5$, $DIR(X, Y) = 5$, and $DIRT(X, Y) = 5$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y, that holds simultaneously $R(X, Y) = 5$, $DIR(X, Y) = 4$, and $DIRT(X, Y) = 3$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y of the length $2n$, $n \in N$, that $R(X, Y) = 2n$ and a) $DIR(X, Y) = 2$; b) $DIRT(X, Y) = \lceil \frac{n}{2} \rceil$

Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ ($*$) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

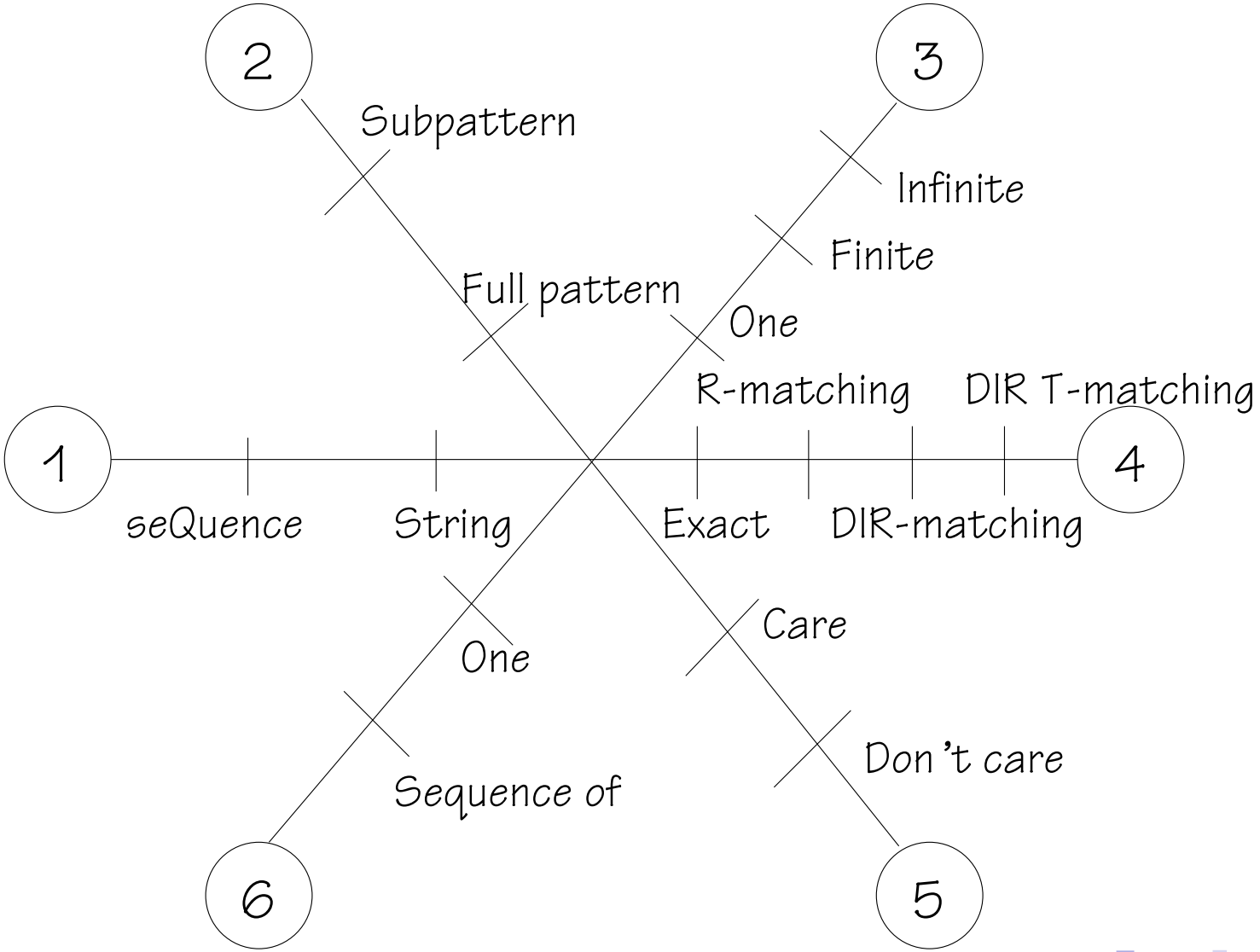
Classification of search problems

Definition: Let $T = t_1 t_2 \dots t_n$ and pattern $P = p_1 p_2 \dots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \dots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

6D classification of search problems [MEH] ([MAR])



6D classification of search problems (cont.)

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

In total $2 \times 2 \times 3 \times 4 \times 2 \times 2 = 192$ search problems classified in a six-dimensional space.

For example, SFO??? denotes all the SE for search of one (entire) string.

For all these problems, we are going to learn how to create NFA for searching.

6D classification of search problems (cont.)

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

In total $2 \times 2 \times 3 \times 4 \times 2 \times 2 = 192$ search problems classified in a six-dimensional space.

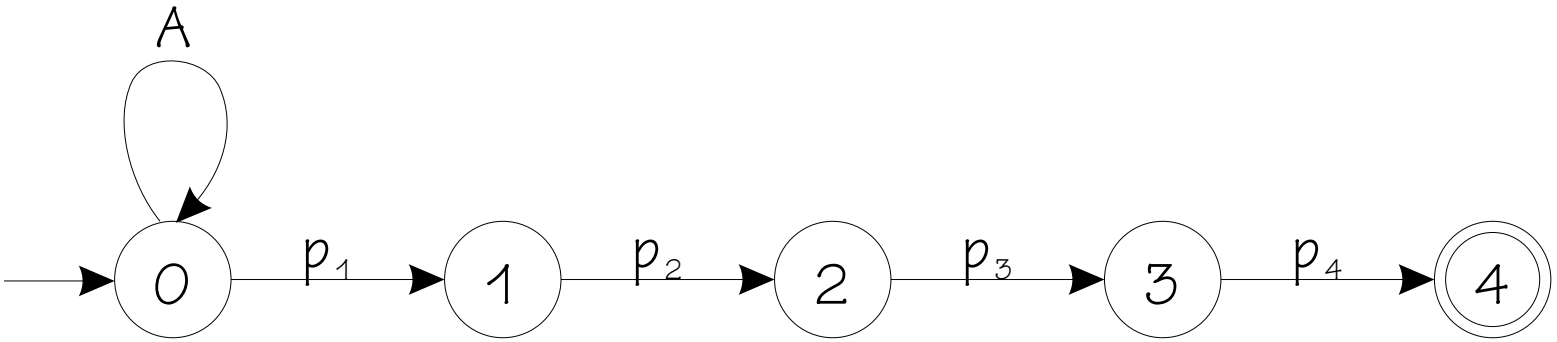
For example, SFO??? denotes all the SE for search of one (entire) string.

For all these problems, we are going to learn how to create NFA for searching.

Examples of SE creation

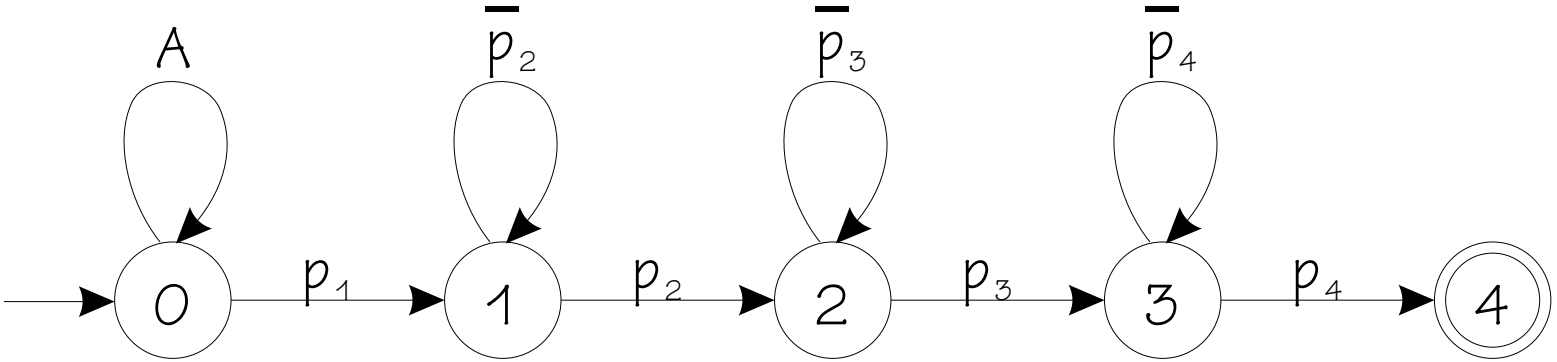
Example: let $P = p_1 p_2 p_3 \ldots p_m$, $m = 4$, A is any character of Σ.

NFA for SFOECO:



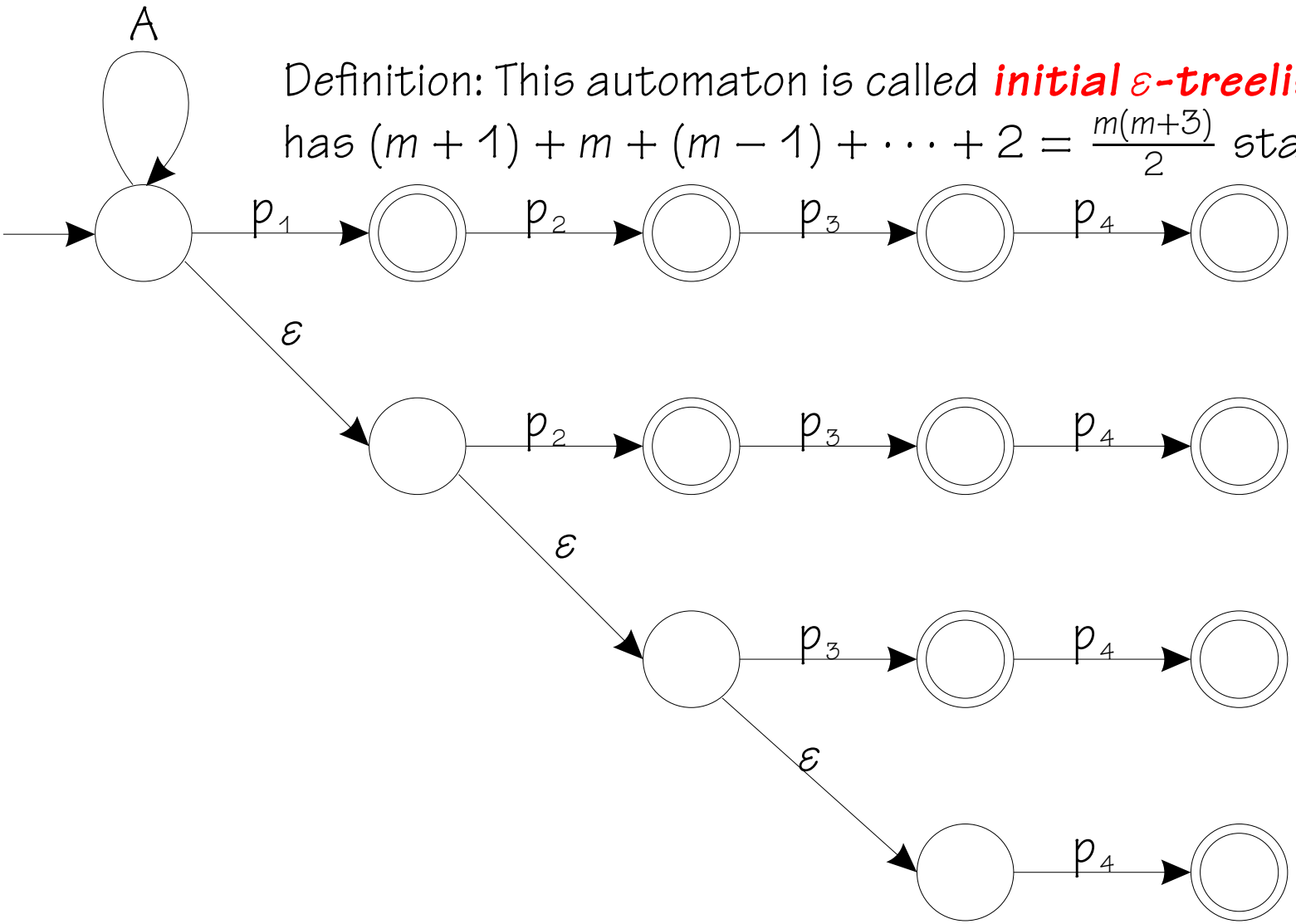
Search for a sequence of characters

Example: NFA for QFOECO (se**Q**uence):



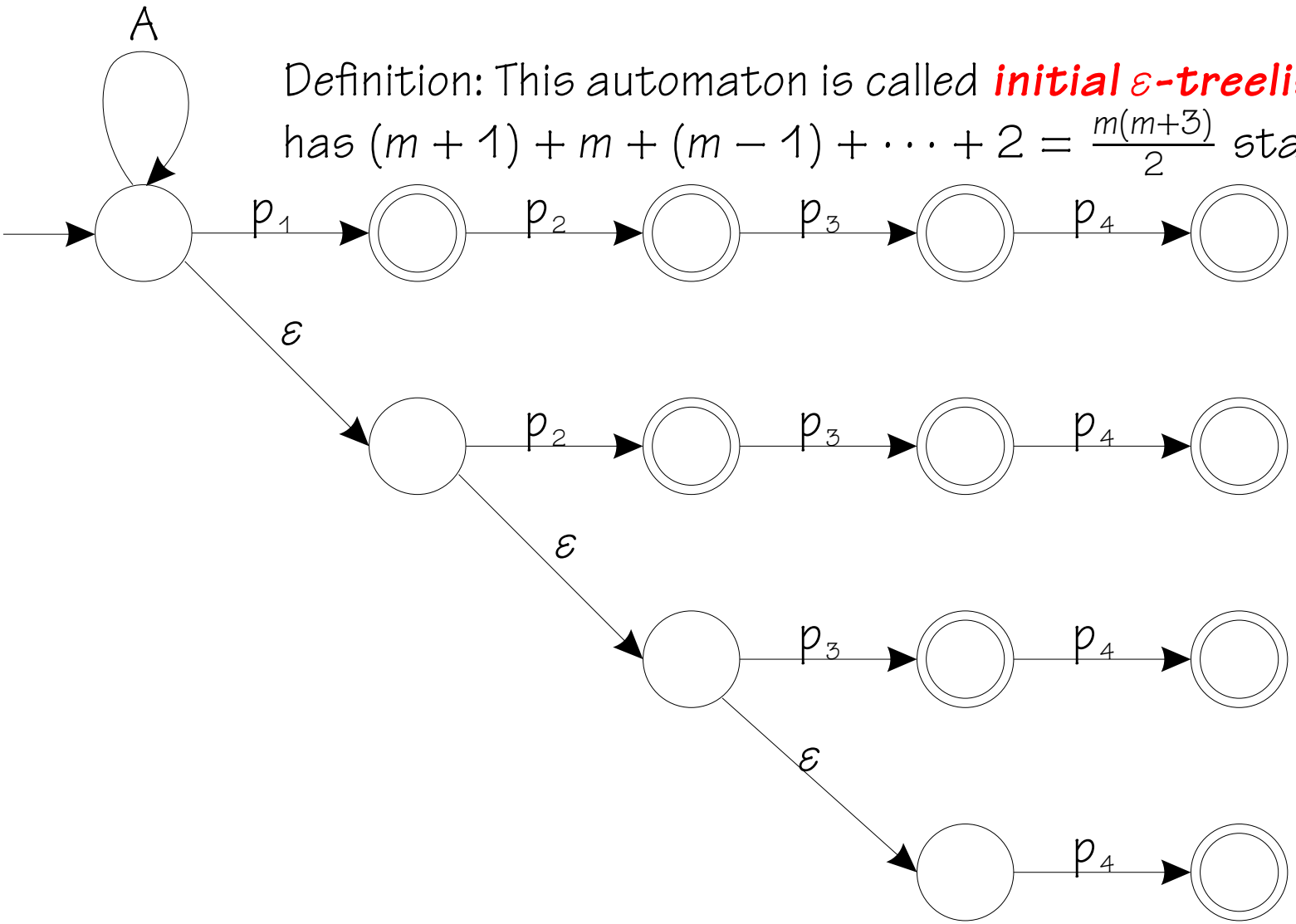
$\overline{p}$ is any character of Σ except for p. Automaton has $m + 1$ states for a pattern of the length m.

Search for a substring: NFA for SSOECO



Definition: This automaton is called ***initial ε-treelis*** and has $(m+1)+m+(m-1)+\cdots+2=\frac{m(m+3)}{2}$ states.

Search for a substring: NFA for SSOECO



Definition: This automaton is called ***initial ε-treelis*** and has $(m+1)+m+(m-1)+\cdots+2=\frac{m(m+3)}{2}$ states.

Search for a subsequence

Example: NFA for QSOECO is similar, we just add some cycles for non-matching characters and ε transitions to all the existing forward transitions (or we concatenate the automaton m-times).

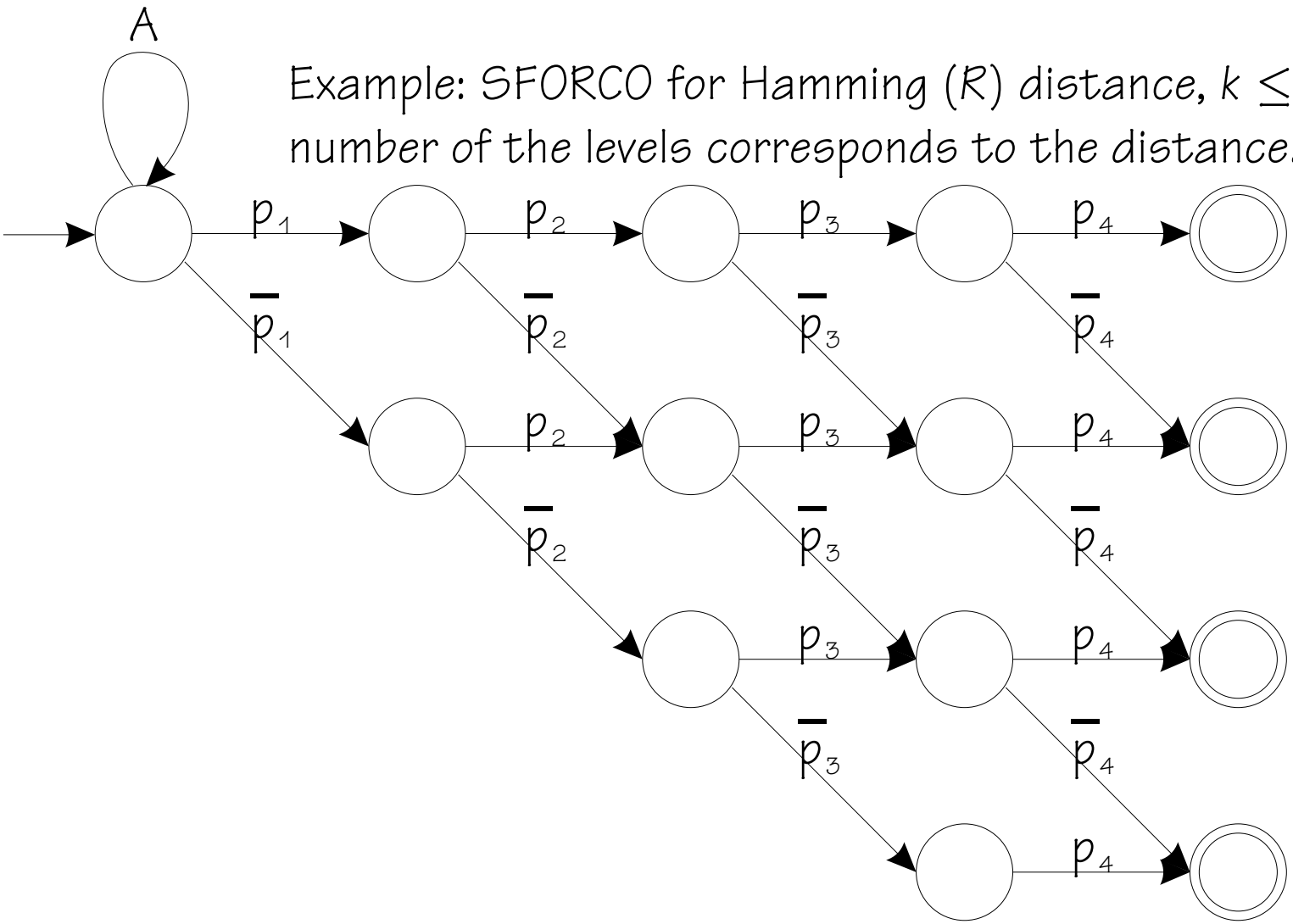
Definition: Automaton for QSOECO is called **ε-treelis**.

Search for a subsequence

Example: NFA for QSOECO is similar, we just add some cycles for non-matching characters and ε transitions to all the existing forward transitions (or we concatenate the automaton m-times).

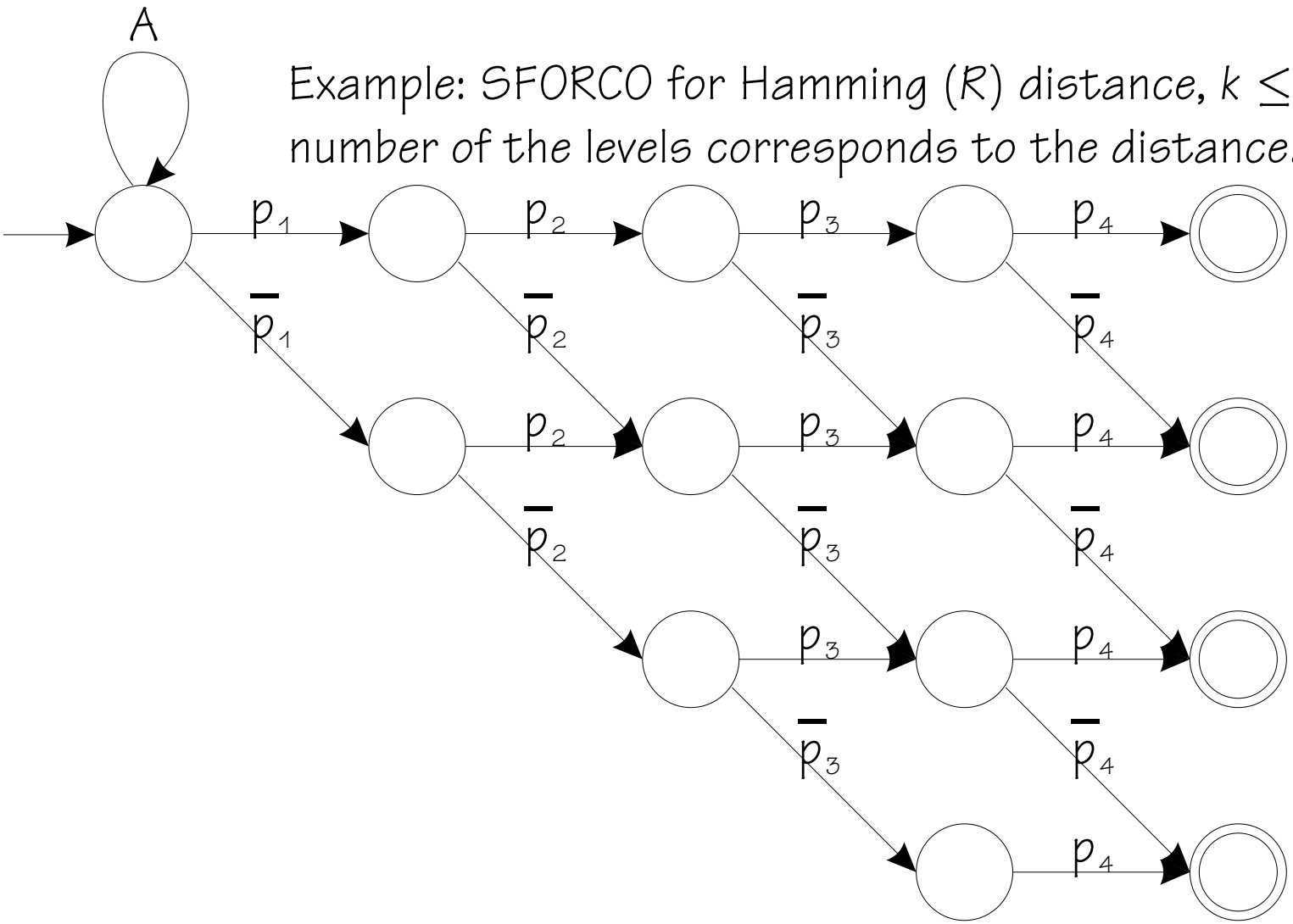
Definition: Automaton for QSOECO is called **ε-treelis**.

Proximity search of SFORCO



Example: SFORCO for Hamming (R) distance, $k \leq 3$: the number of the levels corresponds to the distance.

Proximity search of SFORCO



Example: SFORCO for Hamming (R) distance, $k \leq 3$: the number of the levels corresponds to the distance.

Proximity search of SFORCO

Definition: This automaton is called ***R-treelis***, and has $(m+1)+m+(m-1)+\cdots+(m-k+1) = (k+1)(m+1-\frac{k}{2})$ states.

The number of the level of the final state corresponds to the length of the found string from the pattern.

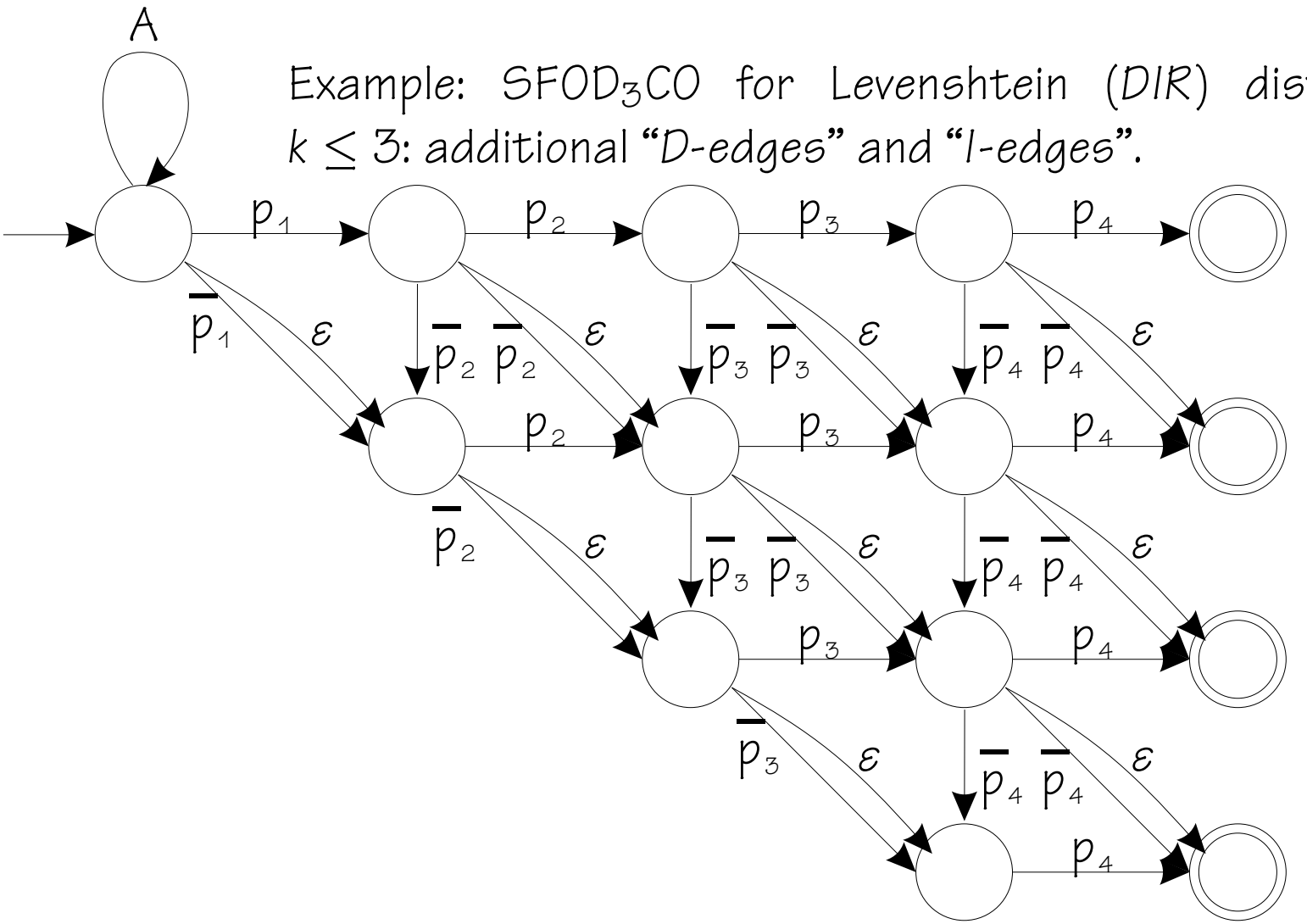
Proximity search of SFORCO

Definition: This automaton is called ***R-treelis***, and has $(m+1)+m+(m-1)+\cdots+(m-k+1) = (k+1)(m+1-\frac{k}{2})$ states.

The number of the level of the final state corresponds to the length of the found string from the pattern.

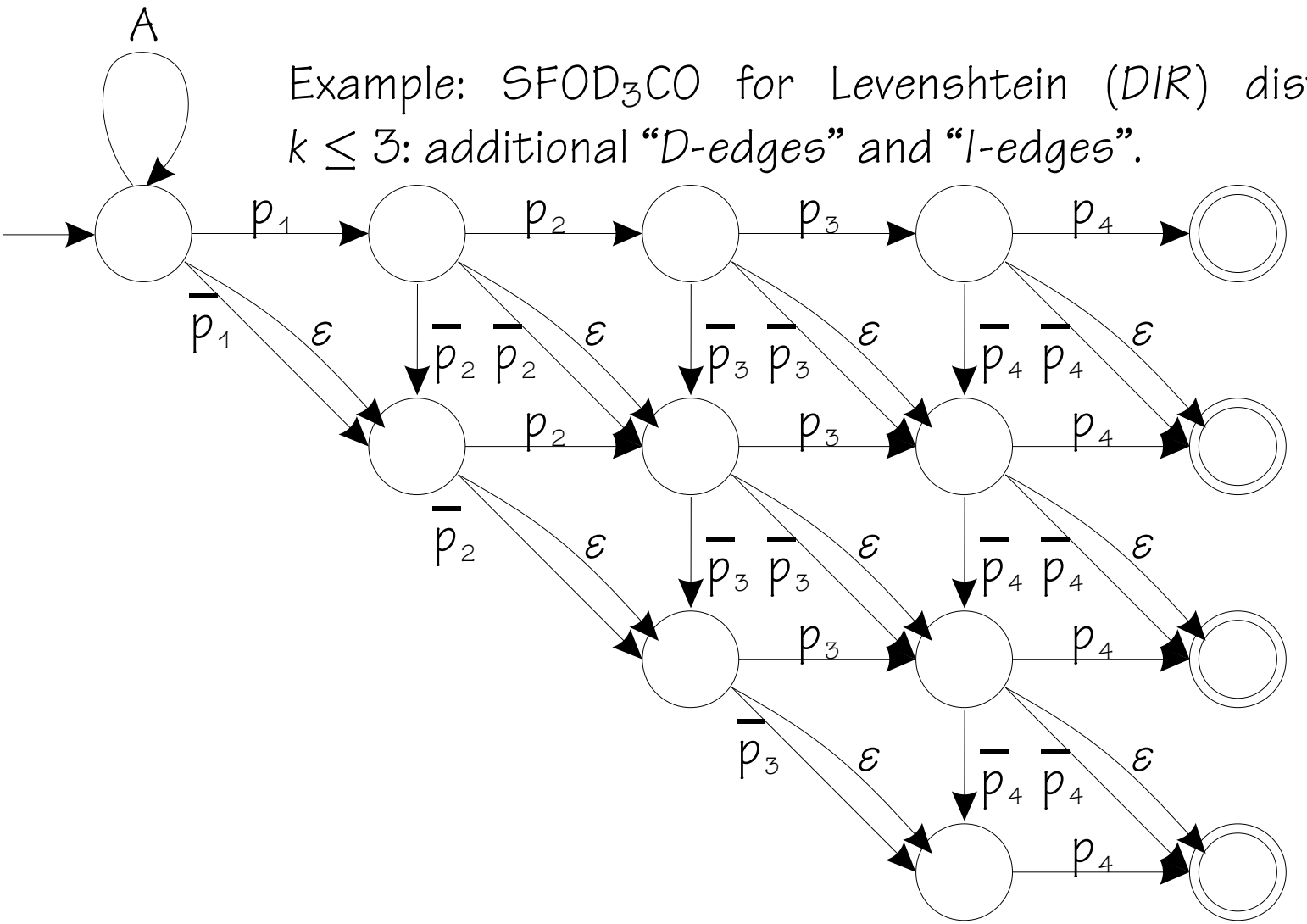
Proximity search of SFODCO for *DIR*-distance

Example: $SFOD_3CO$ for Levenshtein (*DIR*) distance, $k \leq 3$: additional "*D*-edges" and "*I*-edges".



Proximity search of SFODCO for *DIR*-distance

Example: $SFOD_3CO$ for Levenshtein (*DIR*) distance, $k \leq 3$: additional "*D*-edges" and "*I*-edges".



SFOGCO

For the *DIRT*-distance, we add more new states to the SFODCO automaton that correspond to the operation of transposition and also the corresponding pair of edges for every transposition.

Animation program by Mr. Pojer for the discussed search automata is available for download from the course web page and is also installed in B311.

Simulation of NFA or determinisation? A hybrid approach.

The Prague Stringology Club and its conference series: see `http://www.stringology.org/`.

SFOGCO

For the *DIRT*-distance, we add more new states to the SFODCO automaton that correspond to the operation of transposition and also the corresponding pair of edges for every transposition.

Animation program by Mr. Pojer for the discussed search automata is available for download from the course web page and is also installed in B311.

Simulation of NFA or determinisation? A hybrid approach.

The Prague Stringology Club and its conference series: see http://www.stringology.org/.

SFOGCO

For the DIRT-distance, we add more new states to the SFODCO automaton that correspond to the operation of transposition and also the corresponding pair of edges for every transposition.

Animation program by Mr. Pojer for the discussed search automata is available for download from the course web page and is also installed in B311.

Simulation of NFA or determinisation? A hybrid approach.

The Prague Stringology Club and its conference series: see http://www.stringology.org/.

SFOGCO

For the DIRT-distance, we add more new states to the SFODCO automaton that correspond to the operation of transposition and also the corresponding pair of edges for every transposition.

Animation program by Mr. Pojer for the discussed search automata is available for download from the course web page and is also installed in B311.

Simulation of NFA or determinisation? A hybrid approach.

The Prague Stringology Club and its conference series: see `http://www.stringology.org/`.

Outline (Week five)

1. Searching with text preprocessing; indexing methods.
2. Methods of indexing.
3. Automatic indexing, thesaurus construction.
4. Ways of index implementation.

Outline (Week five)

① Searching with text preprocessing; indexing methods.

② Methods of indexing.

③ Automatic indexing, thesaurus construction.

④ Ways of index implementation.

Outline (Week five)

① Searching with text preprocessing; indexing methods.

② Methods of indexing.

③ Automatic indexing, thesaurus construction.

④ Ways of index implementation.

Outline (Week five)

① Searching with text preprocessing; indexing methods.

② Methods of indexing.

③ Automatic indexing, thesaurus construction.

④ Ways of index implementation.

Part II

Indexing Methods

Searching with text preprocessing

Large amount of texts? The text preprocessing!

- Index, indexing methods, indexing file, indexsequential file.
- Hierarchical structuring of text, ***tagging*** of text, ***hypertext***.
- Questions of word list storing (***lexicon***) and occurrence (hit) list storing, their updating.

Searching with text preprocessing

- granularity of the index items: *document – paragraph – sentence – word*

	word1	word2	word3	word4
doc1	1	1	0	1
doc2	0	1	1	1
doc3	1	0	1	1

- ***inverted file***, transposition

	doc1	doc2	doc3
word1	1	0	1
word2	1	1	0
word3	0	1	1
word4	1	1	1

Index searching

- Word order (primary key) in index $\longrightarrow$ ***binary search***
  Time complexity of one word searching in index: n index length, V pattern length
  $O(V \times \log_2(n))$
- searching for k words, pattern $p = v_1, \ldots, v_k$
  $k \ll n \Rightarrow$ ***repeated binary search***
  s average pattern length, complexity?
  $O(s \times k \times \log_2 n)$
- As long as k and i are comparable: ***double dictionary method***.
- ***Hashing***.

However the speed $O(n)$ even $O(\log n)$ isn't usually sufficient, $O(1)$ is needed.

Implementation of indexing systems I

An appropriate choice of data structures and algorithms is for the index implementation crucial.

- Use of inverted file:

word1	1	0	1
word2	1	1	0
word3	0	1	1
word4	1	1	1

- Use of document list:

word1	1, 3
word2	1, 2
word3	2, 3
word4	1, 2, 3

- Coordinate system with pointers has 2 parts: a dictionary with pointers to the document list and a linked list of pointers to documents.

Indexing methods

- manual vs. automatic, pros/cons
- ***stop-list*** (words with grammatical meaning – conjunctions, prepositions, ...)
  1. not-driven
  2. driven (a special dictionary of words: indexing language assessment) – ***pass-list***, thesaurus.
- synonyms and related words.
- inflective languages: creating of registry with language support – ***lemmatization***.

Text analysis – choice of words for index

Frequency of word occurrences is for document identification significant.

English frequency dictionary:

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

- ***Zipf's law*** (principle of least resistance)
  $order \times frequency \cong constant$
- ***Cumulative proportion of used words*** $CPW = \frac{\sum_{order=1}^{N} frequency_{order}}{text\ words\ count}$
- The rule 20–80: 20 % of the most frequent words make 80 % of text [MEL, fig. 4.19].

Automatic indexing method

Automatic indexing method is based on word significance derivation from word frequencies (cf. Collins-Cobuild dictionary); words with low and high frequency are cut out:
INPUT: n documents
OUTPUT: a list of words suitable for an index creation

1. We calculate a frequency $FREQ_{ik}$ for every document $i \in \langle 1, n \rangle$ and every word $k \in \langle 1, K \rangle$ [K is a count of different words in all documents].
2. We calculate $TOTFREQ_k = \sum_{i=1}^{n} FREQ_{ik}$.
3. We create a frequency dictionary for the words $k \in \langle 1, K \rangle$.
4. We set down a threshold for an exclusion of very frequent words.
5. We set down a threshold for an exclusion of words with a low frequency.
6. We insert the remaining words to the index.

Questions of threshold determination [MEL, fig. 4.20].