Text Search

- Dictionary NFA and text search
- Dictionary DFA and text search
- Hamming distance and Dynamic Programming?
- Levenshtein distance and Dynamic Programming
- Resume
- Boyer-Moore text search approach

Literature: Borivoj Melichar, Jan Holub, Tomas Polcar
TEXT SEARCHING ALGORITHMS VOLUME I.
CTU, FEE, Nov 2005

Finite language

Dictionary over an alphabet A is a finite set of strings (patterns) from A*.
Dictionary automaton searches the text for any pattern in the given dictionary.

Recycle older knowledge

1. Dictionary is a finite language.
2. Each finite language is a regular language.
3. Each regular language can be described by a regular expression.
4. Any language described by a regular expression can be searched for in any text using appropriate NFA/DFA.

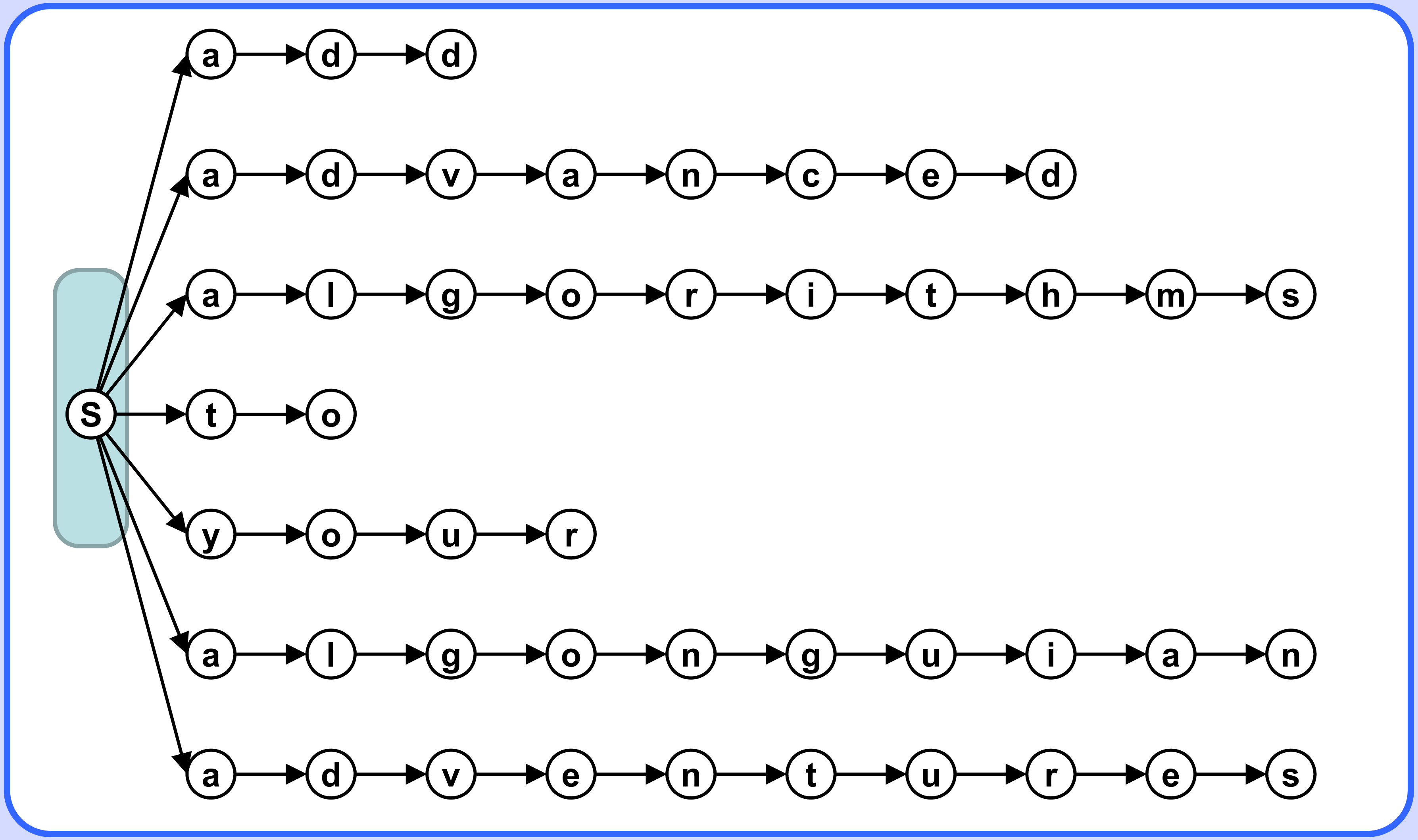
Example

Alphabet
A = {a, c, d, e, g, h, i, l, m, n, o, q, r, s, t, u, v, y}
Dictionary
D = {"add", "advanced", "algorithms", "to", "your", "algonqiuan", "adventures"}

The *Algonquian* are one of the most populous and widespread North American native language groups.

Finite language

Merge repeatedly into a single state any two states A and B such that path from S to A and from S to B are of equal length and contain equal sequence of transition labels. You may find e.g. BFS/DFS to be useful.



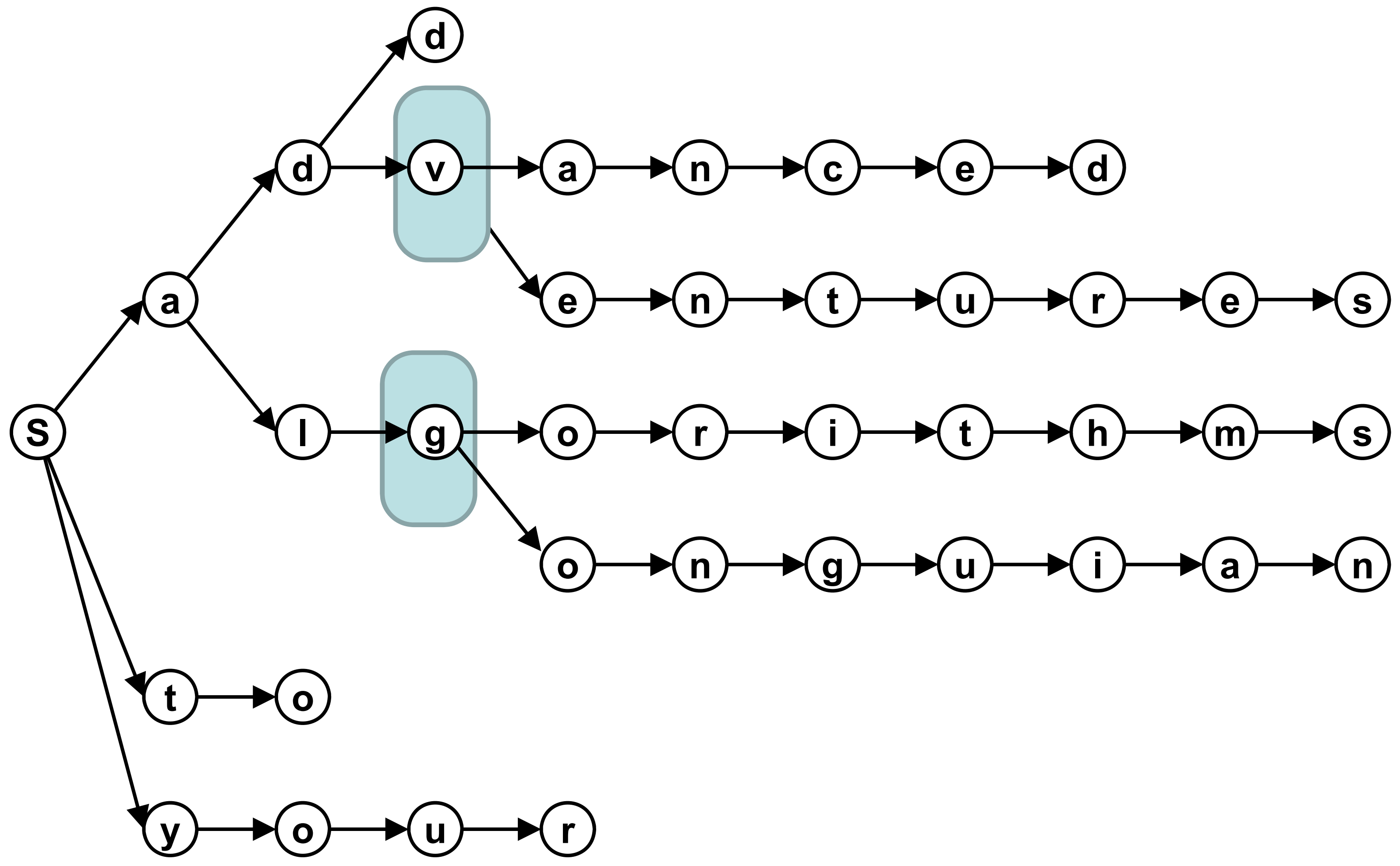
Finite language

$$\begin{tikzcd}
 & & d \arrow[r] & d & & & & & & & \\
 & & d \arrow[r] & v \arrow[r] & a \arrow[r] & n \arrow[r] & c \arrow[r] & e \arrow[r] & d & & \\
 & a \arrow[uur] \arrow[ur] \arrow[r] \arrow[dr] \arrow[ddr] & l \arrow[r] & g \arrow[r] & o \arrow[r] & r \arrow[r] & i \arrow[r] & t \arrow[r] & h \arrow[r] & m \arrow[r] & s \\
S \arrow[ur] \arrow[ddr] \arrow[dddr] & & l \arrow[r] & g \arrow[r] & o \arrow[r] & n \arrow[r] & g \arrow[r] & u \arrow[r] & i \arrow[r] & a \arrow[r] & n \\
 & & d \arrow[r] & v \arrow[r] & e \arrow[r] & n \arrow[r] & t \arrow[r] & u \arrow[r] & r \arrow[r] & e \arrow[r] & s \\
 & t \arrow[r] & o & & & & & & & & \\
 & y \arrow[r] & o \arrow[r] & u \arrow[r] & r & & & & & &
\end{tikzcd}$$

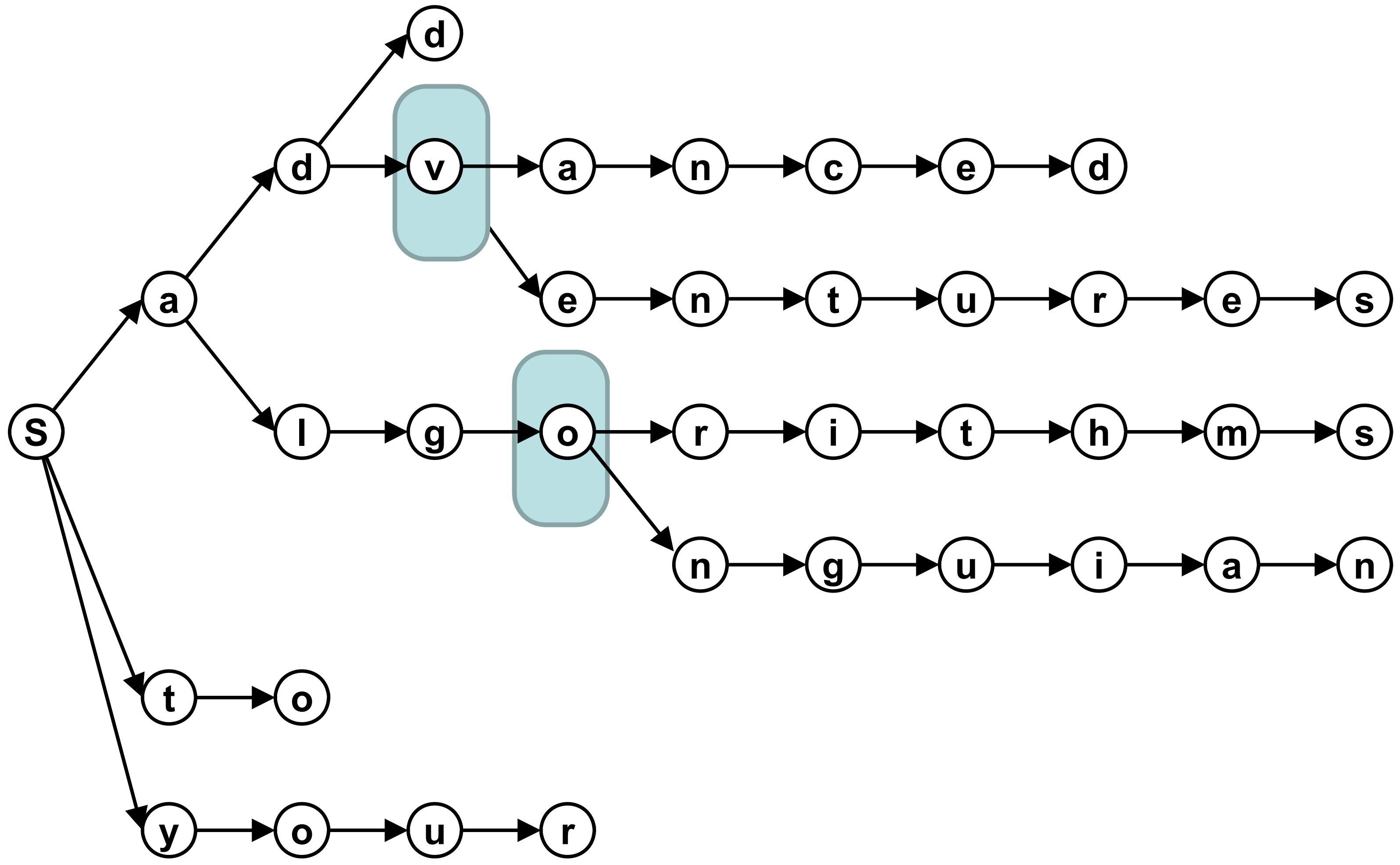
Finite language

$$\begin{tikzcd}
 & & & d & & & & & & & \\
 & & d \arrow[ur] \arrow[r] \arrow[dr] & v \arrow[r] & a \arrow[r] & n \arrow[r] & c \arrow[r] & e \arrow[r] & d & & \\
 & a \arrow[ur] \arrow[dr] & & v \arrow[r] & e \arrow[r] & n \arrow[r] & t \arrow[r] & u \arrow[r] & r \arrow[r] & e \arrow[r] & s \\
S \arrow[ur] \arrow[ddr] \arrow[dddr] & & l \arrow[r] \arrow[dr] & g \arrow[r] & o \arrow[r] & r \arrow[r] & i \arrow[r] & t \arrow[r] & h \arrow[r] & m \arrow[r] & s \\
 & & & g \arrow[r] & o \arrow[r] & n \arrow[r] & g \arrow[r] & u \arrow[r] & i \arrow[r] & a \arrow[r] & n \\
 & t \arrow[r] & o & & & & & & & & \\
 & y \arrow[r] & o \arrow[r] & u \arrow[r] & r & & & & & &
\end{tikzcd}$$

Finite language

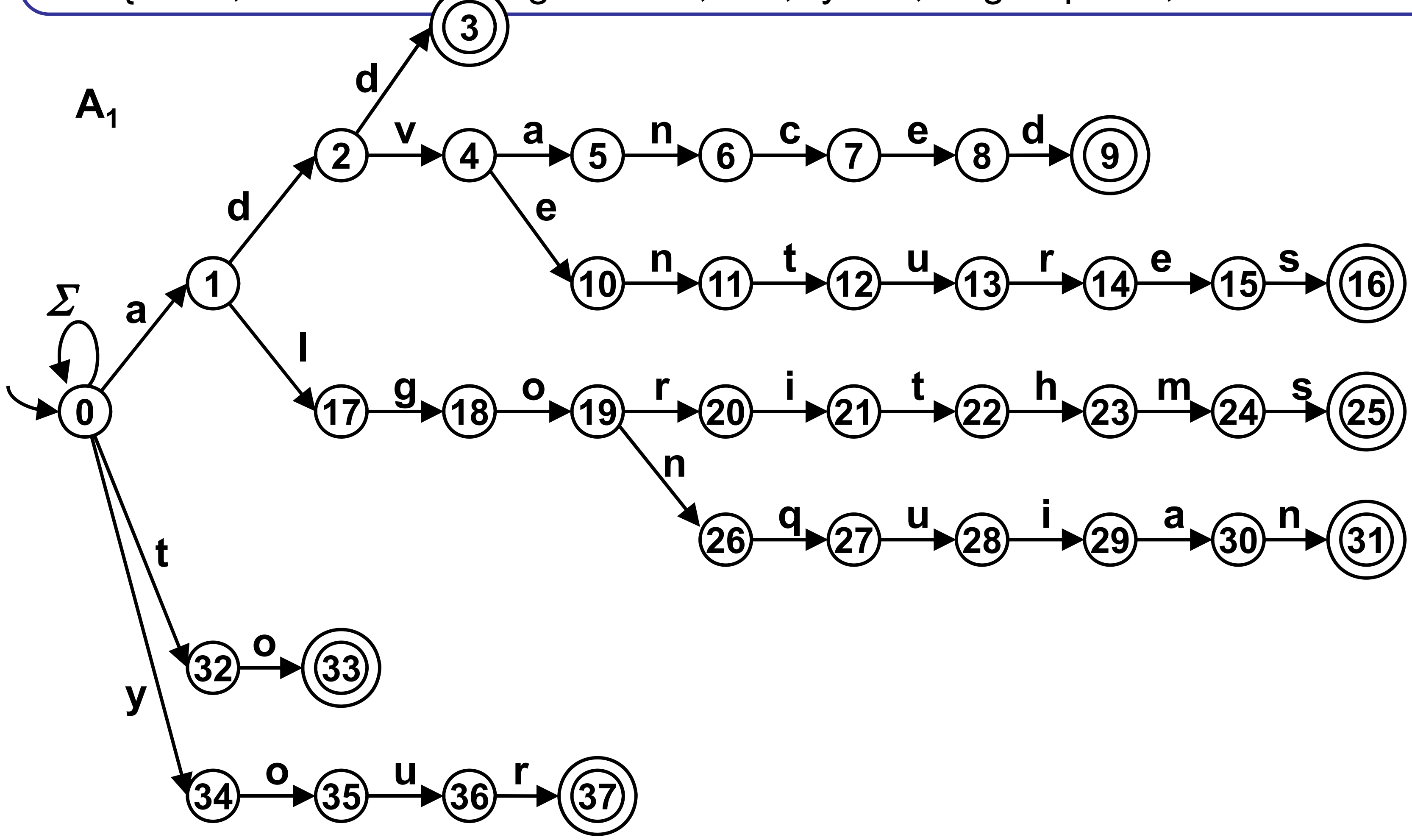


Finite language

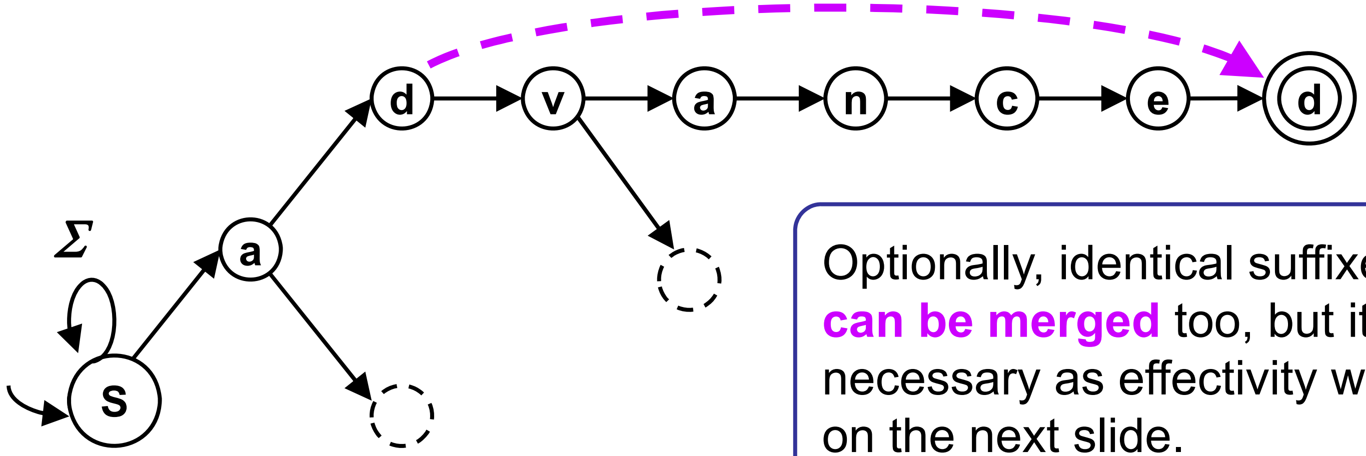


Dictionary NFA

Search NFA for dictionary
D = {"add", "advanced", "algorithms", "to", "your", "algonqiuan", "adventures"}

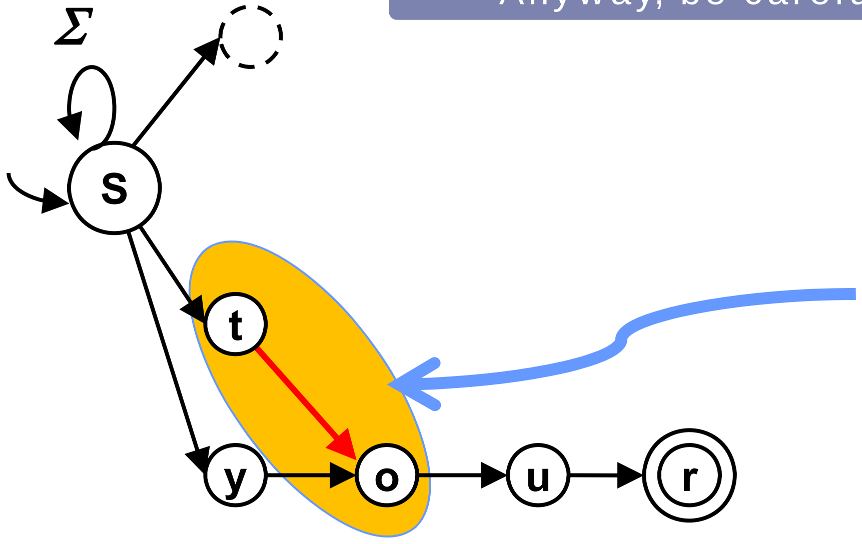


Small Optimization



Optionally, identical suffixes **can be merged** too, but it is not necessary as effectivity will be granted on the next slide.

Anyway, be careful.



This is a wrong construction.
It would incorrectly add word "tour" to the dictionary.

Dictionary DFA

The transition diagram of a dictionary NFA, like A_1 in the previous example, is a **directed tree** with the start state **in the root**.
The **only loop** is the self-loop **in the start state** labeled by the whole alpahbet.
This NFA has an usefull property:

Effectivity

Transforming dictionary NFA of this shape to DFA does not increase the number of states.

Example

The transition diagram of the resulting DFA has 38 states (same as NFA) and 684 transitions. It would not fit nicely into one slide, therefore we present only the transition table... :

Homework: Draw it!

Dictionary DFA

	a	c	d	e	g	h	i	l	m	n	o	q	r	s	t	u	v	y	
0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,1	0,1	0	0,2	0	0	0	0	0,17	0	0	0	0	0	0	0,32	0	0	0,34	
0,32	0,1	0	0	0	0	0	0	0	0	0	0,33	0	0	0	0,32	0	0	0,34	
0,34	0,1	0	0	0	0	0	0	0	0	0,35	0	0	0	0	0,32	0	0	0,34	
0,2	0,1	0	0,3	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0,4	0,34	
0,17	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,33	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,35	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0,36	0	0,34	
0,3	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,4	0,1,5	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,18	0,1	0	0	0	0	0	0	0	0	0	0,19	0	0	0	0,32	0	0	0,34	
0,36	0,1	0	0	0	0	0	0	0	0	0	0	0	0,37	0	0,32	0	0	0,34	
0,1,5	0,1	0	0	0,2	0	0	0	0,17	0	0,6	0	0	0	0	0,32	0	0	0,34	

Transition table of DFA A_2 equivalent to dictionary NFA A_1.

Continue...

Dictionary DFA

... continued

	a	c	d	e	g	h	i	l	m	n	o	q	r	s	t	u	v	y	
0,10	0,1	0	0	0	0	0	0	0	0	0,11	0	0	0	0	0,32	0	0	0,34	
0,19	0,1	0	0	0	0	0	0	0	0	0,26	0	0	0,20	0	0,32	0	0	0,34	
0,37	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,6	0,1	0,7	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,11	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,12, 32	0	0	0,34	
0,26	0,1	0	0	0	0	0	0	0	0	0	0	0,27	0	0	0,32	0	0	0,34	
0,20	0,1	0	0	0	0	0	0,21	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,7	0,1	0	0	0,8	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,12, 32	0,1	0	0	0	0	0	0	0	0	0	0,33	0	0	0	0,32	0,13	0	0,34	
0,27	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0,28	0	0,34	
0,21	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,22, 32	0	0	0,34	
0,8	0,1	0	0,9	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,13	0,1	0	0	0	0	0	0	0	0	0	0	0	0,14	0	0,32	0	0	0,34	

continue...

Dictionary DFA

... continued

	a	c	d	e	g	h	i	l	m	n	o	q	r	s	t	u	v	y	
0,28	0,1	0	0	0	0	0	0,29	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,22,32	0,1	0	0	0	0	0,23	0	0	0	0	0,33	0	0	0	0,32	0	0	0,34	
0,9	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,14	0,1	0	0	0,15	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,23	0,1	0	0	0	0	0	0	0	0,24	0	0	0	0	0	0,32	0	0	0,34	
0,29	0,1,30	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,15	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0,16	0,32	0	0	0,34	
0,24	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0,25	0,32	0	0	0,34	
0,1,30	0,1	0	0,2	0	0	0	0	0,17	0	0,31	0	0	0	0	0,32	0	0	0,34	
0,16	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,25	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,31	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F

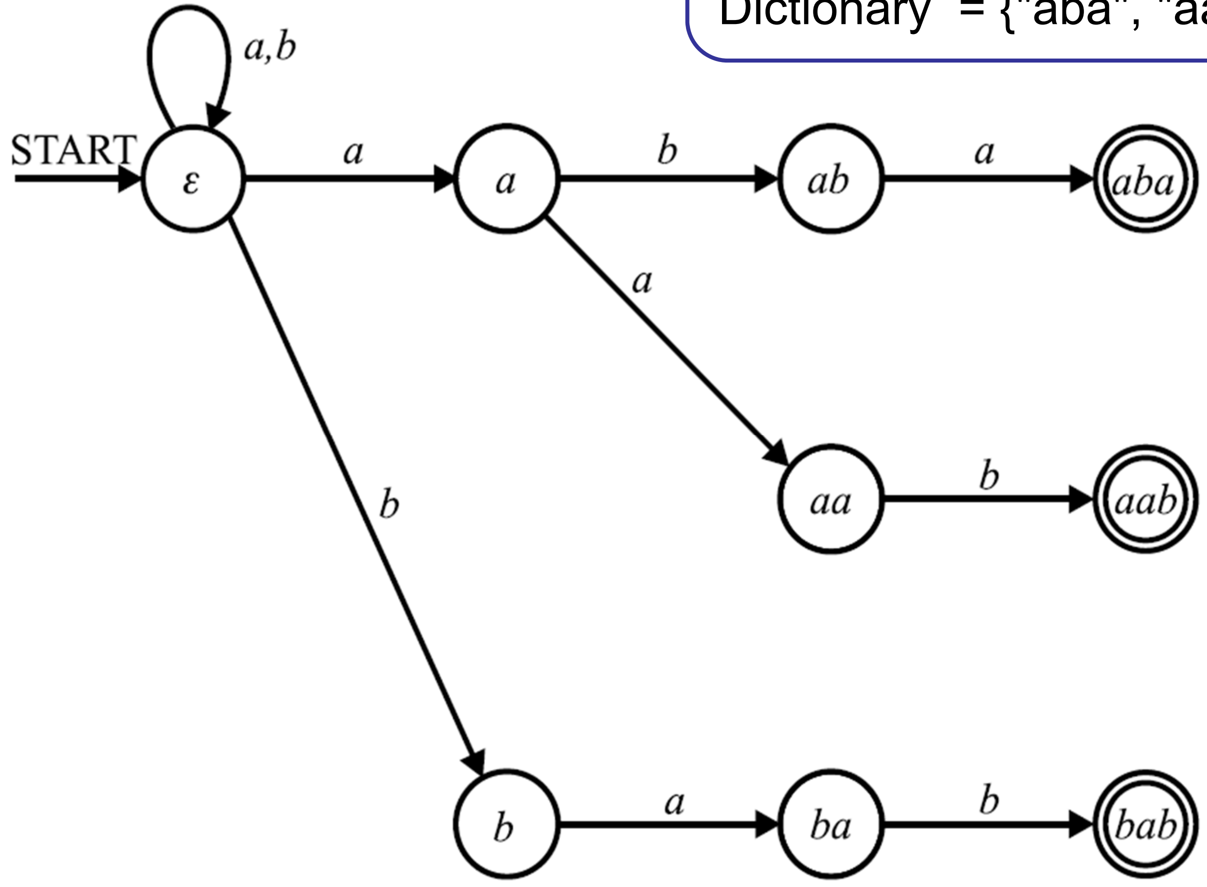
... finished.

Dictionary DFA

Example of a dictionary automaton whose transition diagram fits to one slide.

NFA

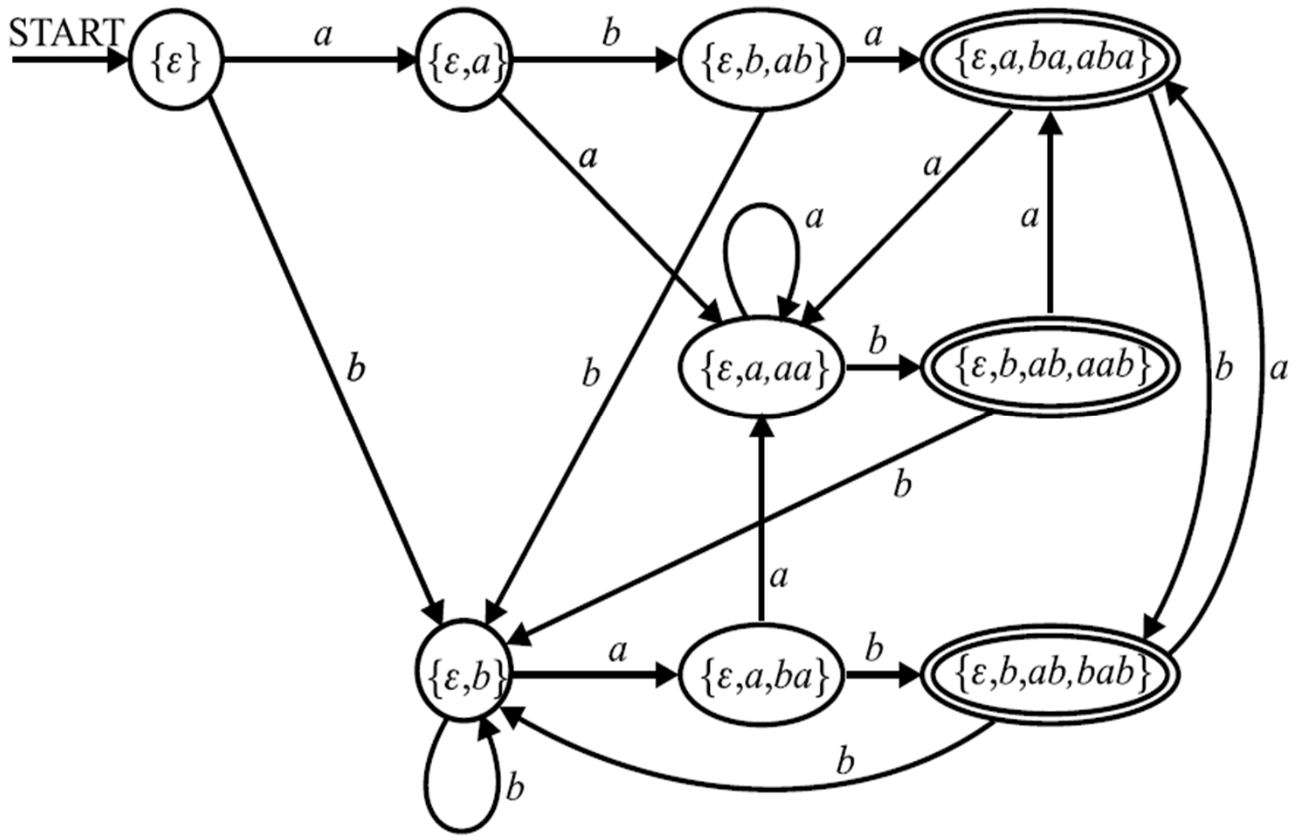
Alphabet = {a, b}
Dictionary = {"aba", "aab", "bab"}



Dictionary DFA

Alphabet = {a, b}
Dictionary = {"aba", "aab", "bab"}

DFA



DP approach to text search considering Hamming distance

Alphabet {a,b,c,d}, pattern P: adbbca, text T: adcabcaabadbbca.
For each each alignment P with T determine
Hamming distance between P and t[k-m+1], t[k-m+2] ..., t[k]

T:	t[1]	...	t[k-m+1]	...	t[k-1]	t[k]	...	t[n]
P:			p[1]	...	p[m-1]	p[m]		

Method

Let pattern P be p[1], p[2], ..., p[m], let text T be t[1], t[2], ..., t[n].

Create dynamic programming table D[m+1][n+1], whose elements d[i][k] are defined as follows:

```
1. d[0][k] = 0                                    // for k = 0, ..., n

2. if (p[i] == t[k])  d[i][k] = d[i-1][k-1]
   else               d[i][k] = d[i-1][k-1] +1   // for 1 ≤ i ≤ k, i ≤ m, k ≤ n,
```

Fill the table row by row. Element d[m][k] holds the Hamming distance of P from the substring t[k-m+1], t[k-m+2] ..., t[k].

Hamming Distance

Alphabet {a,b,c,d}, pattern P: adbbca, text T: adcabcaabadbbca.

D	-	a	d	c	a	b	c	a	a	b	a	d	b	b	c	a
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	-	0	1	1	0	1	1	0	0	1	0	1	1	1	1	0
d	-	-	0	2	2	1	2	2	1	1	2	0	2	2	2	2
b	-	-	-	1	3	2	2	3	3	1	2	3	0	2	3	3
b	-	-	-	-	2	3	3	3	4	3	2	3	3	0	3	4
c	-	-	-	-	-	3	3	4	4	5	4	3	4	4	0	4
a	-	-	-	-	-	-	-	3	4	5	5	5	4	5	5	0

🟦 Highligted cells represent a match between the text and the pattern.

Though it looks scientifically advanced,
it is, in fact, only a basic naive approach :-).

Each diagonal corresponds to some alignment of pattern with text
where mismatches in this alignment are counted one by one.

DP approach to text search considering Levenshtein distance

Let pattern P be p[1], p[2], ..., p[m], let text T be t[1], t[2], ..., t[n].

Create dynamic programming table D[m+1][n+1], whose elements d[i][k] are defined as follows:

```
1. d[i][0] = i;  d[0][k]  = 0,   for i = 0, ..., m, k = 1, ..., n

2.  // d[i][k] is computed using the information about
    //  the minimum possible number of applications of operations
    // delete, insert, rewrite to the strings shorter by one last character
    // and followed by at most one edit operation
   for 1 ≤ i ≤ m, 1 ≤ k ≤ n:
      d[i][k] = minimum of (
               d[i−1][k] +1,                                // delete p[i]
               if( i < m )  d[i][k−1] + 1,                  // insert after p[i]
               d[i−1][k−1]  +  (p[i] == t[k]) ? 0 : 1 )     // leave or rewrite p[i]
```

Fill the table row by row. The cell d[m][k] contains the minimum Levenshtein distance of P from the substring $S_{x,k}$ = t[x], t[x+1], ..., t[k], where $x \in$ { k−m+1−d[m][k], ..., k-m+1+d[m][k] } and the particular value of x is *not known*.

Levenshtein Distance

Alphabet {a,b,c,d}, pattern P: adbbca, text T: adcabcaabadbbca.

D	-	a	d	c	a	b	c	a	a	b	a	d	b	b	c	a
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1	0	1	1	0	1	1	0	0	1	0	1	1	1	1	0
d	2	1	0	1	1	1	2	1	1	1	1	0	1	2	2	1
b	3	2	1	1	2	1	2	2	2	1	2	1	0	1	2	2
b	4	3	2	2	2	2	2	3	3	2	2	2	1	0	1	2
c	5	4	3	2	3	3	2	3	4	3	3	3	2	1	0	1
a	6	5	4	3	2	4	3	2	3	4	3	4	3	2	1	0

▢ Highligted cells represent a match between the text and the pattern.

```
d[i][k] = minimum of (
        d[i−1][k] +1,                                  // delete p[i]
        if( i < m )  d[i][k−1] + 1,                    // insert after p[i]
        d[i−1][k−1]  +  (p[i] == t[k]) ? 0 : 1 )       // leave or rewrite p[i]
```

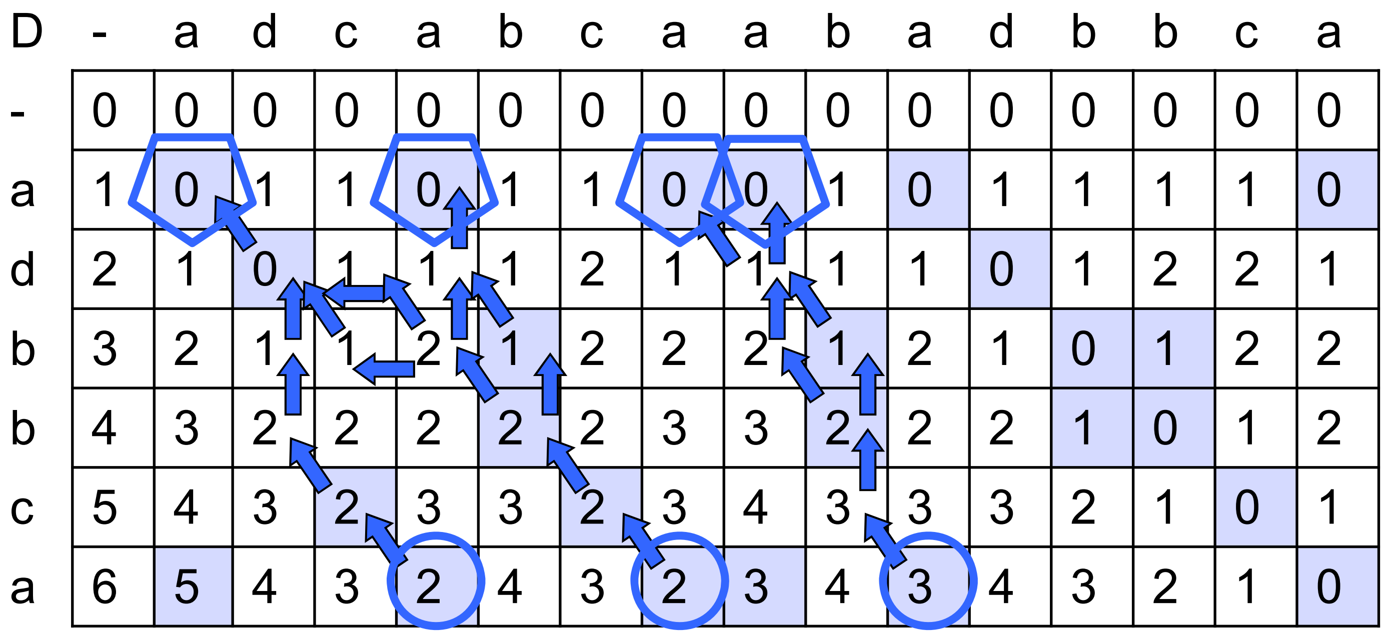
Levenshtein Distance

D	-	a	d	c	a	b	c	a	a	b	a	d	b	b	c	a
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1	0	1	1	0	1	1	0	0	1	0	1	1	1	1	0
d	2	1	0	1	1	1	2	1	1	1	1	0	1	2	2	1
b	3	2	1	1	2	1	2	2	2	1	2	1	0	1	2	2
b	4	3	2	2	2	2	2	3	3	2	2	2	1	0	1	2
c	5	4	3	2	3	3	2	3	4	3	3	3	2	1	0	1
a	6	5	4	3	2	4	3	2	3	4	3	4	3	2	1	0

Challenge

Value d[m][k] registers only the distance of a substring S in the text whose end is aligned with end of P and it is the minimum distance of all such substrings. There is no reference in the DP table to the actual length of S i.e. to its start position. To find string
$S = S_x = t[x], t[x+1] \ldots, t[k]$, where $x \in \{k-m+1-d[m][k], \ldots, k-m+1+d[m][k]\}$
we must consider all values of x and compute Levenshtein distance (S_x, P) for each x separately and choose x which attains minimum.

Levenshtein Distance



D	-	a	d	c	a	b	c	a	a	b	a	d	b	b	c	a
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1	0	1	1	0	1	1	0	0	1	0	1	1	1	1	0
d	2	1	0	1	1	1	2	1	1	1	1	0	1	2	2	1
b	3	2	1	1	2	1	2	2	2	1	2	1	0	1	2	2
b	4	3	2	2	2	2	2	3	3	2	2	2	1	0	1	2
c	5	4	3	2	3	3	2	3	4	3	3	3	2	1	0	1
a	6	5	4	3	2	4	3	2	3	4	3	4	3	2	1	0

▢ Highligted cells represent a match between the text and the pattern.

Only few DAG edges are shown here

d[i][k] = *minimum of* (
   d[i−1][k] +1, // delete p[i]
   if(i < m) d[i][k−1] + 1, // insert after p[i]
   d[i−1][k−1] + (p[i] == t[k]) ? 0 : 1) // leave or rewrite p[i]

a d c a
a d c a b c a
a b c a

a a b a
a b a

In each table cell d[i][k], register as predecessor(s) those of the three neighbour cells d[i−1][k], d[i][k−1], d[i−1][k−1] which may be used to calculate the value in this cell. Each path, in the resulting DAG, from the table bottom row to its 2nd row represent one substring in the text which distance from the pattern is equal to the value in the path root.

Levenshtein distance

Dist("BETELGEUSE","BRUXELLES") = 6

		B	E	T	E	L	G	E	U	S	E
	0	1	2	3	4	5	6	7	8	9	10
B	1	0	1	2	3	4	5	6	7	8	9
R	2	1	1	2	3	4	5	6	7	8	9
U	3	2	2	2	3	4	5	6	6	7	8
X	4	3	3	3	3	4	5	6	7	7	8
E	5	4	3	4	3	4	5	5	6	7	7
L	6	5	4	4	4	3	4	5	6	7	8
L	7	6	5	5	5	4	4	5	6	7	8
E	8	7	6	6	5	5	5	4	5	6	7
S	9	8	7	7	6	6	6	5	5	5	6

Levenshtein distance

Levenshtein distance of strings

Old stuff

```
Dist(A, B) =   |m−n|                                        if n = 0 or m = 0

Dist(A, B) =   1+ min (  Dist(A[1..n−1], B[1..m]),          if  n > 0 and m > 0
                         Dist(A[1..n], B[1..m−1]),              and A[n] ≠ B[m]
                         Dist(A[1..n−1], B[1..m−1]) )

Dist(A, B) =   Dist(A[1..n−1], B[1..m−1]])                     if  n > 0 and m > 0
                                                                and A[n] = B[m]
```

Calculation corresponds to	...	Operation
1+ Dist(A[1..n −1], B[1..m]),	...	**Insert**(A, n−1, B[m]) or **Delete**(B, m)
1+ Dist(A[1..n], B[1..m−1]),	...	**Insert**(B, m−1, A[n]) or **Delete**(A, n)
1+ Dist(A[1..n −1], B[1..m−1])	...	**Rewrite**(A, n, B[m]) or **Rewrite**(B, m, A[n])

Text search considering Levenshtein distance

New stuff

```
d[i][k] = minimum of (
                d[i−1][k] +1,                              //  Delete p[i]
                if (i < m) d[i][k−1] +1,                   // Insert after p[i]
                d[i−1][k−1]  + (p[i] == t[k])?0:1)  )      // leave or Rewrite p[i]
```

Text search using finite automata brings in many possibilities regarding what can be effectively found:

A. Any given exact pattern P.   (e.g. *ababccabc*)

B. Any word of any language specified by a particular DFA or NFA.
(Just add the loop labeled by the whole alphabet to the start state.)

C. Any string which represents some modification of the pattern P:
A string within (or exactly at) a given Hamming distance from P
A string within (or exactly at) a given Levenshtein/edit distance from P.

D. Any of strings in a given (finite) dictionary.

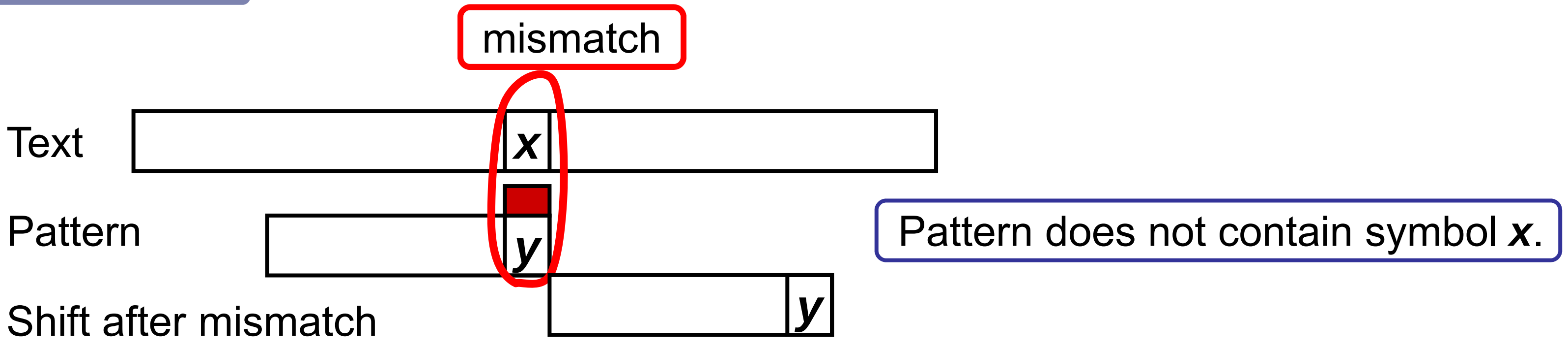
E. Any word of any language described by a regular expression.

F. Any union, intersection, concatenation, iteration of any of cases A. - F.

G. Any string containing any of cases A. - F. as a subsequence.
(Just add the loops labeled by the whole alphabet to all states.)

The idea:
Align the pattern with the text and start matching
backwards **from the end** of the pattern.
When a mismatch occurs there is a chance that the pattern may be shifted forward by many positions and sometimes by the whole pattern length.

Ideal case

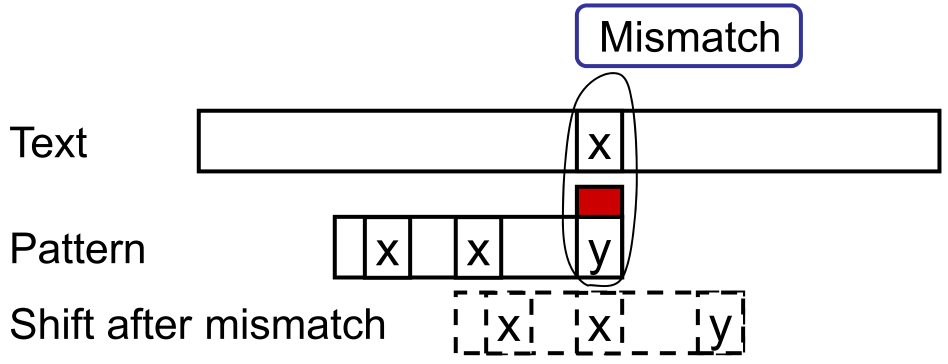


The longer is the pattern the more effective is the search.
(The bigger the data the faster the algorithm, quite an unusual situation...)

Mismatch at the last position of the pattern.

Bad Character Shift table (BCS)
When the last symbol of pattern (y) is mismatched with symbol **x** in the text shift the pattern to the right to match the first occurrence (from the end) of **x** in the pattern with **x** in the text.
When the pattern does not contain **x** shift it by its whole length.
BCS is indexed by all symbols of alphabet.
For each symbol in the pattern it contains the symbol's minimum distance from the end of the pattern. If the symbol is not in the pattern then the table entry is equal to the pattern length.

Example



Text

B	C	C	F	A	B	B	E	C

Pattern

F	A	B	B	E

BCS

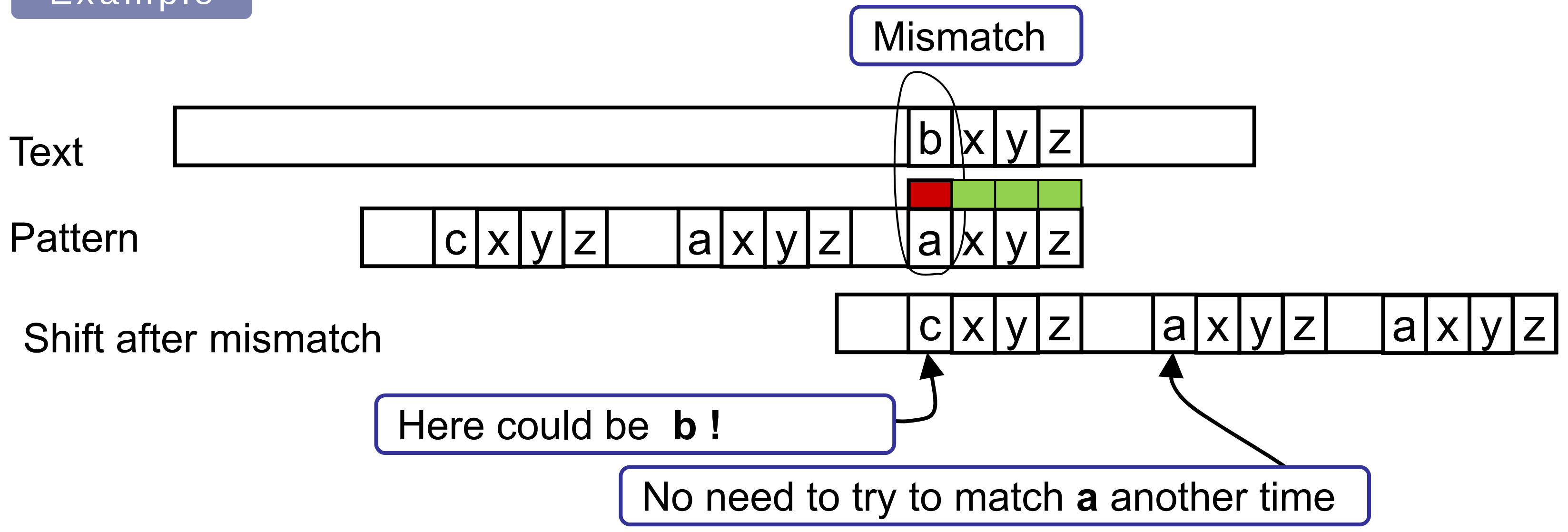
A	B	C	D	E	F
3	1	5	5	0	4

Mismatch after partial match at the end of the pattern.

When a suffix S of the pattern matches the text and the symbol **x** immediately preceding S mismatches the text then there are three cases:

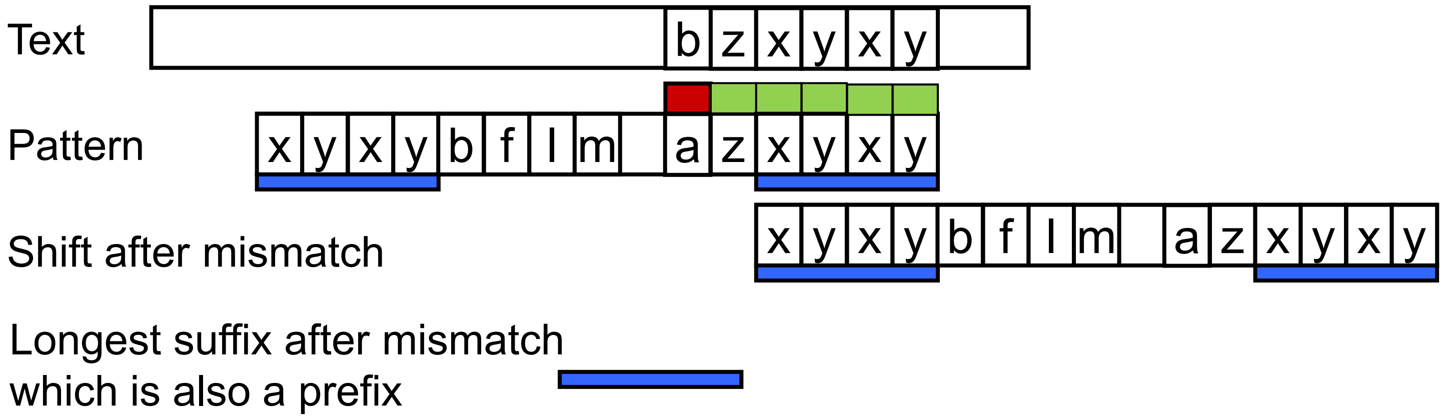
1. The suffix S occurs more times in the pattern and the other occurrence is not immediately preceded by x. In this case, shift the pattern so that the nearest described instance of S matches the text again at the same position. That is, shift the pattern by the distance between these occurrences of suffix S.

Example



2. There is a suffix W whose length does not exceed the length of S and W is also a prefix of the pattern. Take the longest possible W and denote its occurrence at the beginning of the pattern by Q.
Then shift the pattern by the distance between Q and W.

Example


Text
b z x y x y
Pattern
x y x y b f l m a z x y x y
Shift after mismatch
x y x y b f l m a z x y x y
Longest suffix after mismatch
which is also a prefix

3. Neither case 1. nor case 2. happens. Then shift the pattern by its whole length.

Example is unnecessary

Text Search

The shift can be calculated for all three cases :
Take suffix S as a separate string and align it with its original position in the pattern. Then keep shifting S to to the left until one of the cases 1., 2., 3. is detected (at least 3. must happen after some time).
Register the distance between the current and the original position of S.

Good Suffix Shift (GS) table contains the shift values for all suffixes S.

Example

Pattern

A	D	B	A	C	B	A	C	B	A

Pattern length: 10

Positions indexed from 1,
0 represents shift after complete match.

Apply case 2. after complete match

GS

position	mismatches	suffix	shift
9	B	A	9
8	C	BA	6
7	A	CBA	9
6	B	ACBA	9
5	C	BACBA	3
4	A	CBACBA	9
3	B	ACBACBA	9
2	D	BACBACBA	9
1	A	DBACBACBA	10
0	-	ADBACBACBA	9

Example

Pattern

P	O	V	A	L	O	V	A	L

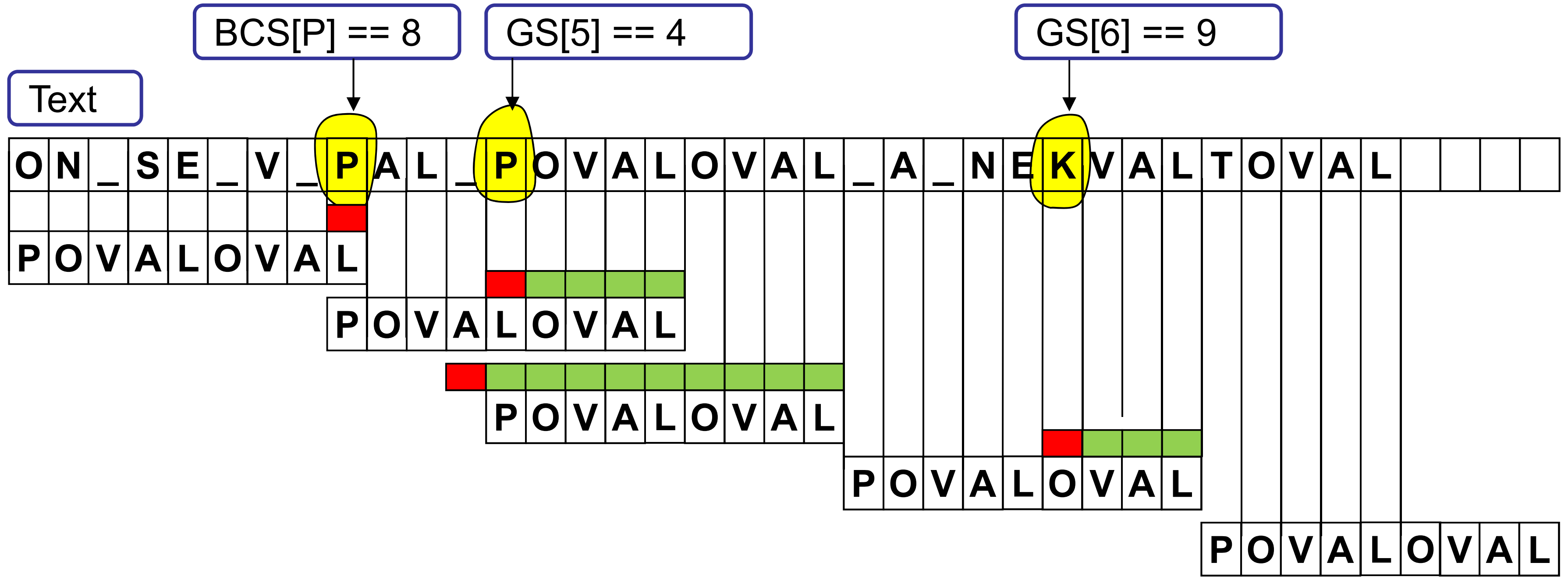
BCS

_	A	E	K	L	N	O	P	S	T	V
9	1	9	9	4	9	3	8	9	9	2

GS

0	1	2	3	4	5	6	7	8	9
9	9	9	9	9	4	9	9	9	-
	P	O	V	A	L	O	V	A	L

Search progress


BCS[P] == 8
GS[5] == 4
GS[6] == 9
Text
O N _ S E _ V _ P A L _ P O V A L O V A L _ A _ N E K V A L T O V A L
P O V A L O V A L
P O V A L O V A L
P O V A L O V A L
P O V A L O V A L
P O V A L O V A L