Text Search

Dictionary NFA and text search

Dictionary DFA and text search

Hamming distance and Dynamic Programming?

Levenshtein distance and Dynamic Programming

...

Resume

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

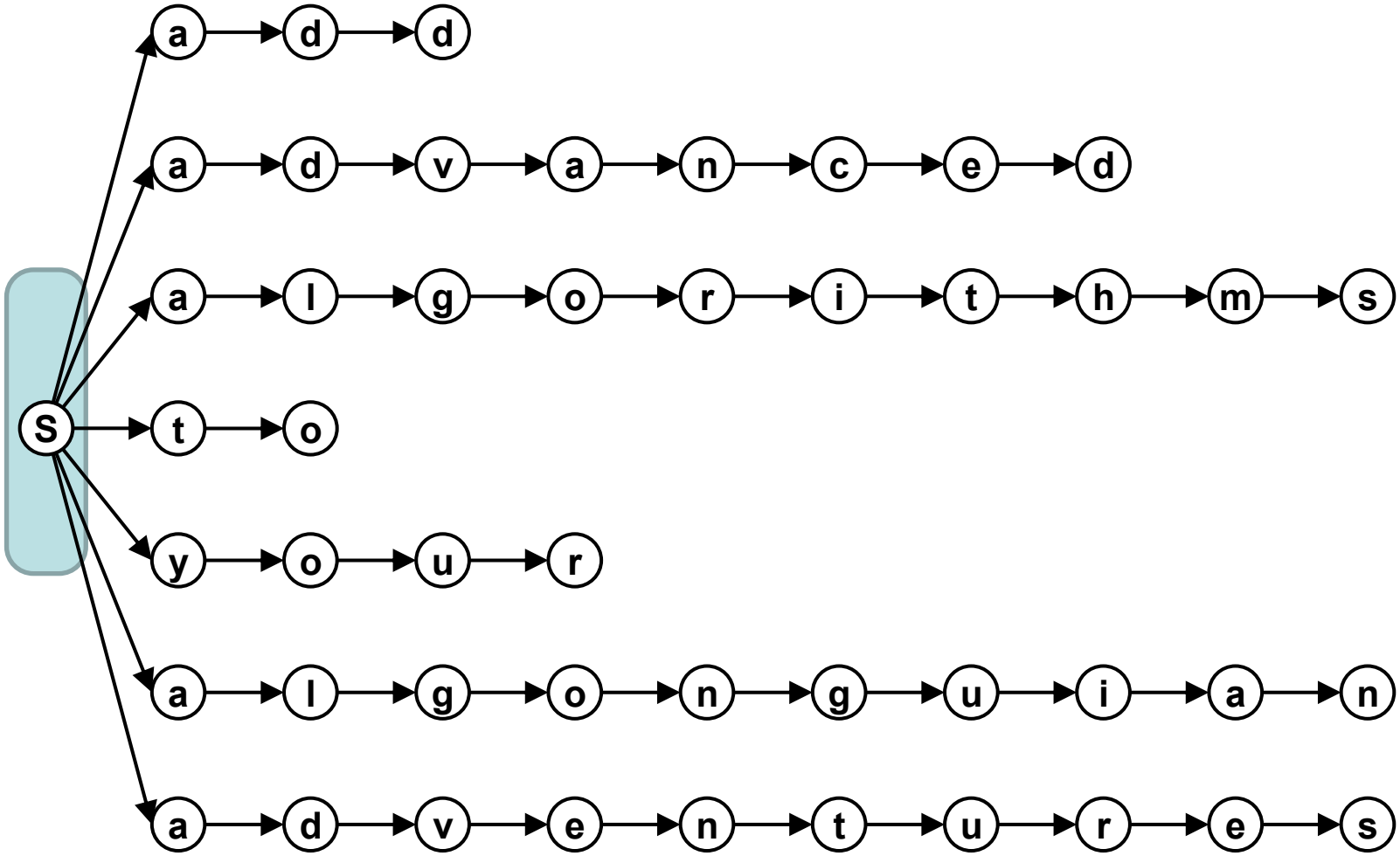
1. Dictionary is a finite language.
2. Each finite language is a regular language.
3. Each regular language can be described by a regular expression.
4. Any language described by a regular expression can be searched for in any text using appropriate NFA/DFA.

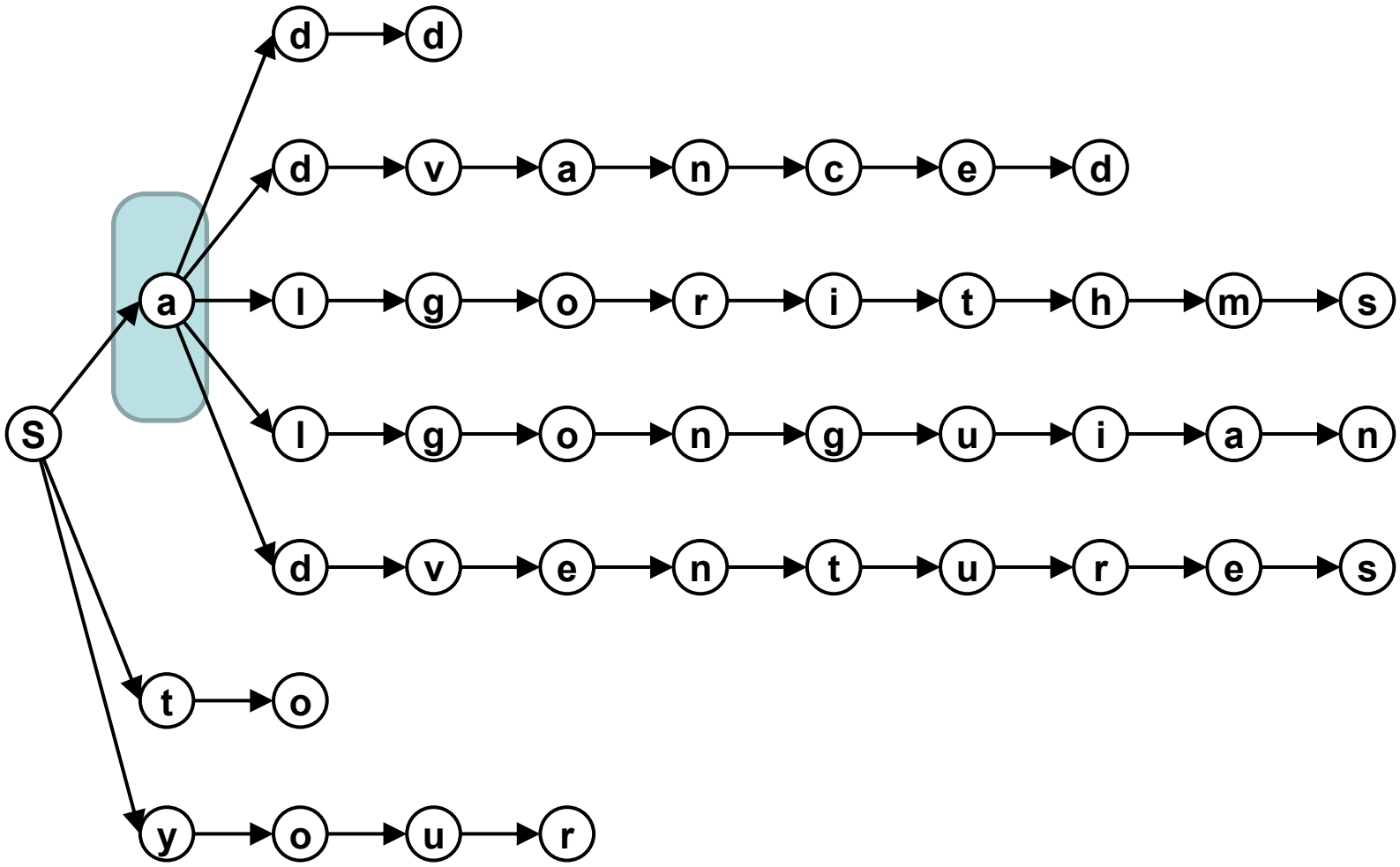
Example

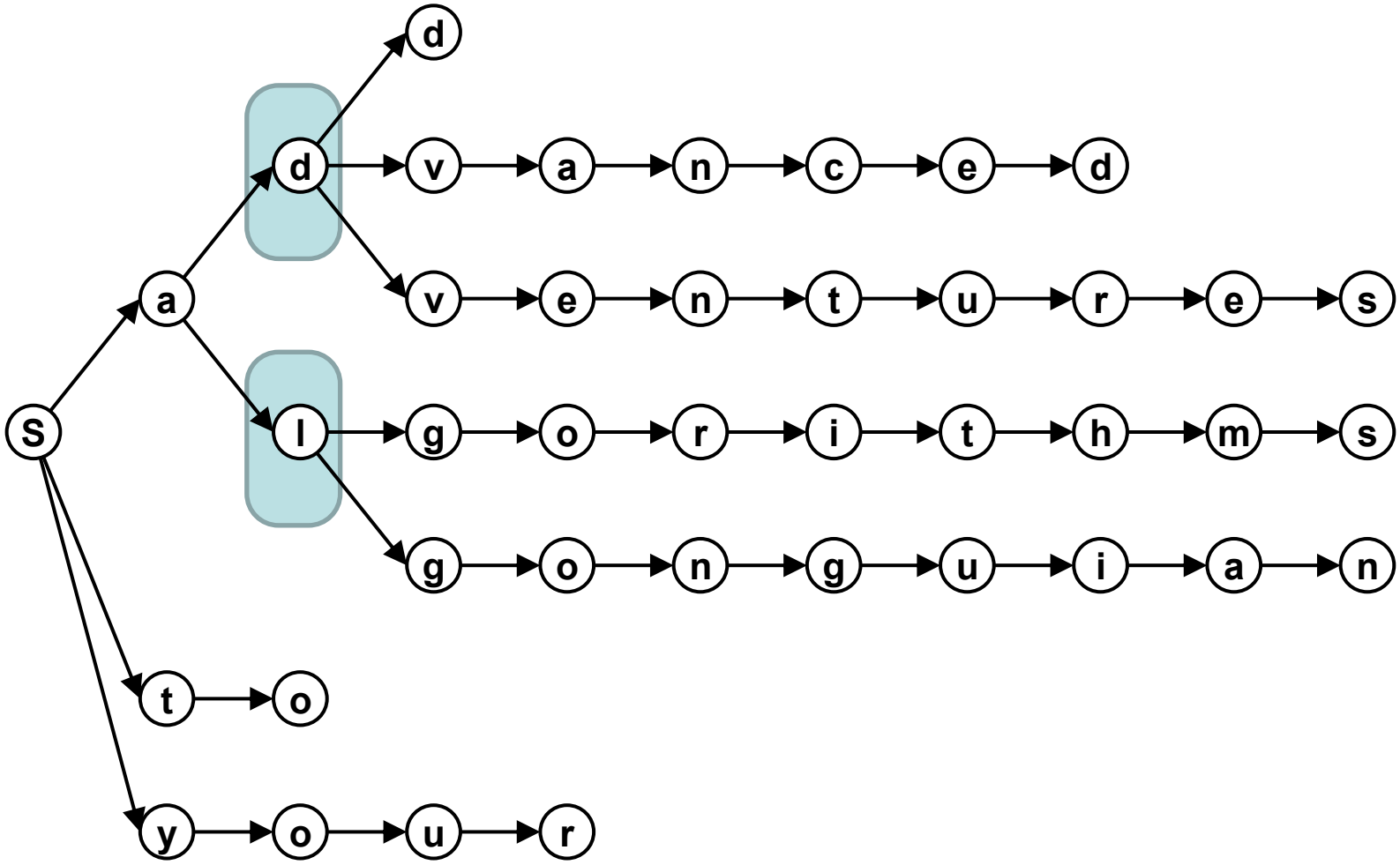
Alphabet
A = {a, c, d, e, g, h, i, l, m, n, o, q, r, s, t, u, v, y}
Dictionary
D = {"add", "advanced", "algorithms", "to", "your", "algonqiuan", "adventures"}

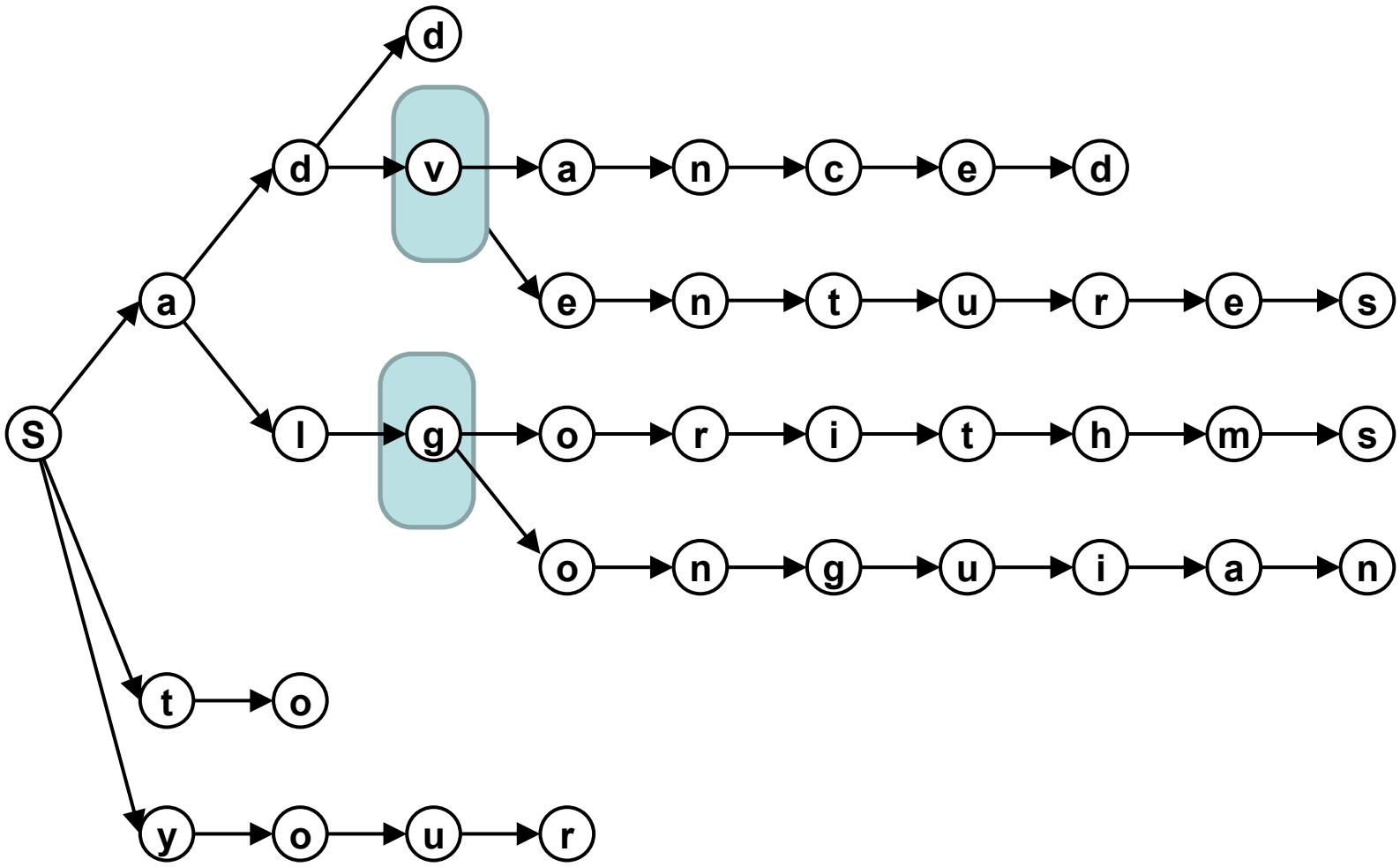
The *Algonquian* are one of the most populous and widespread North American native language groups.

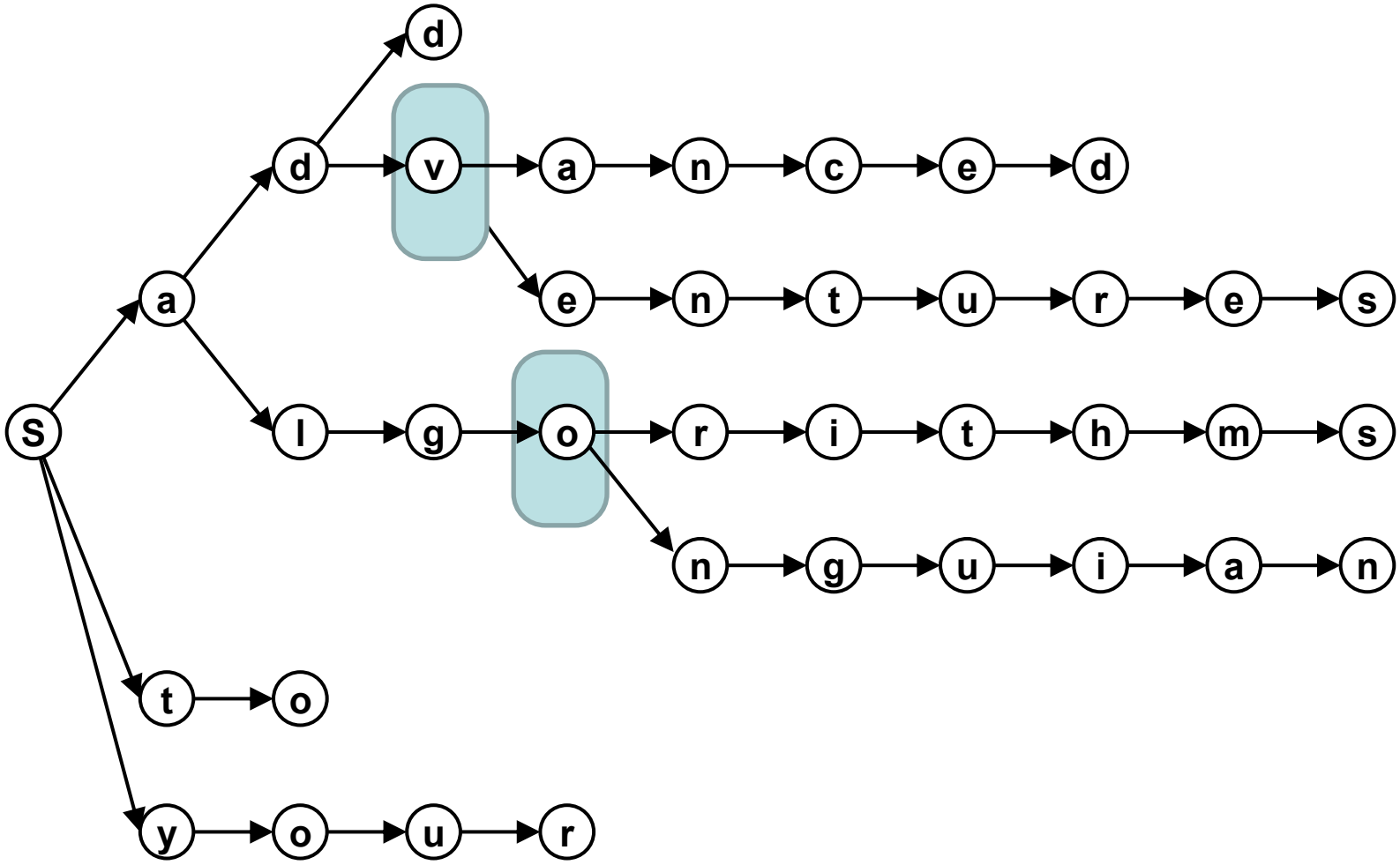
Merge repeatedly into a single state any two states A and B such that path from S to A and from S to B are of equal length and contain equal sequence of transition labels. You may find e.g. BFS/DFS to be useful.



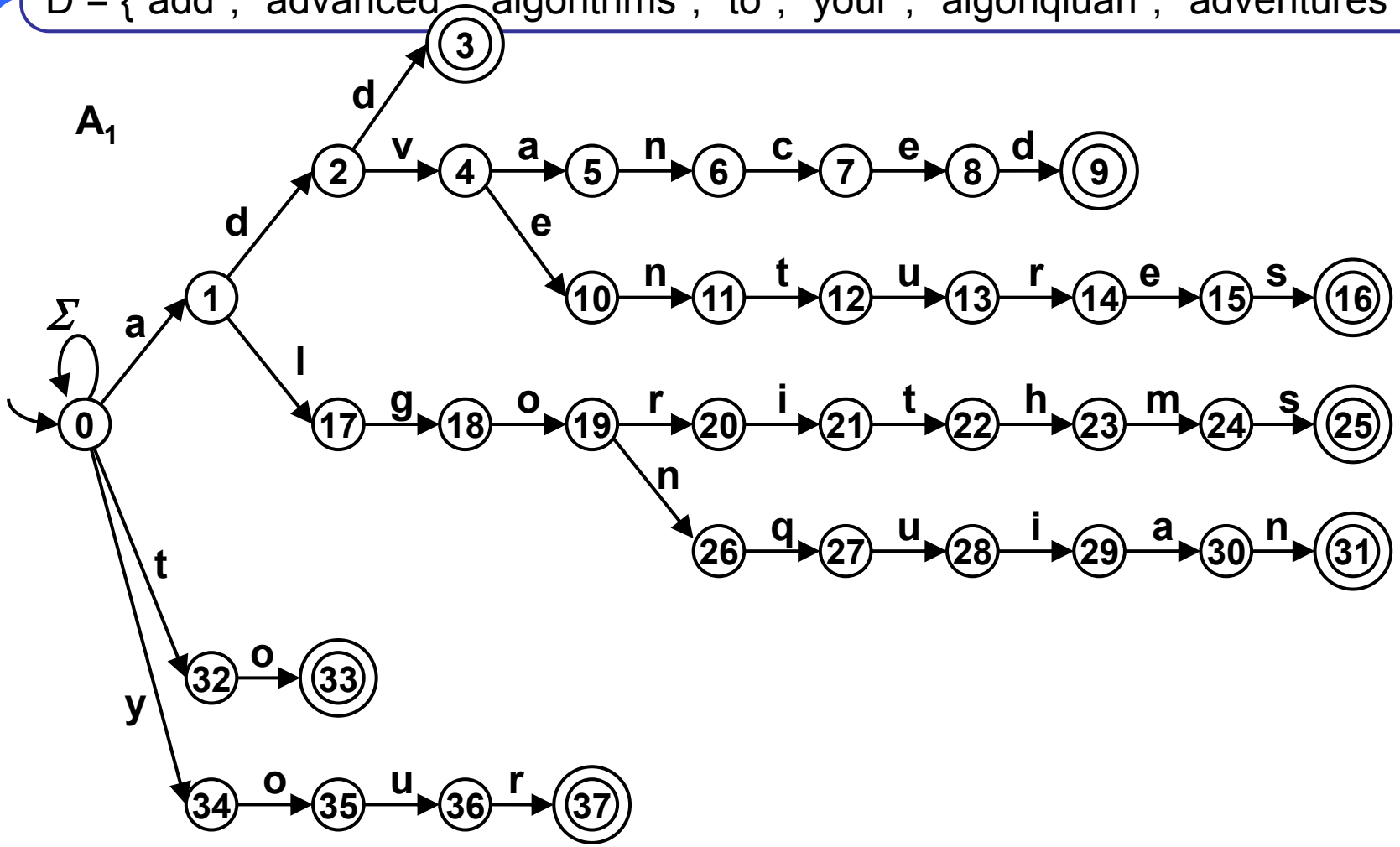

S
a
d
d
d
v
a
n
c
e
d
l
g
o
r
i
t
h
m
s
l
g
o
n
g
u
i
a
n
d
v
e
n
t
u
r
e
s
t
o
y
o
u
r

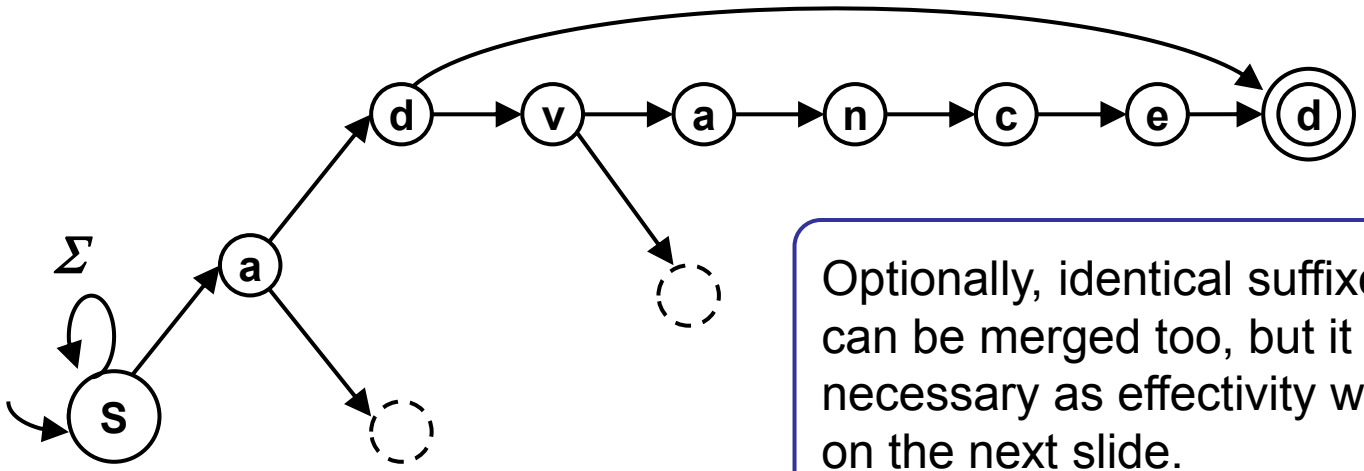

S
a
d
d
v
a
n
c
e
d
v
e
n
t
u
r
e
s
l
g
o
r
i
t
h
m
s
g
o
n
g
u
i
a
n
t
o
y
o
u
r


S
a
d
d
v
a
n
c
e
d
e
n
t
u
r
e
s
l
g
o
r
i
t
h
m
s
o
n
g
u
i
a
n
t
o
y
o
u
r


S
a
d
d
v
a
n
c
e
d
e
n
t
u
r
e
s
l
g
o
r
i
t
h
m
s
n
g
u
i
a
n
t
o
y
o
u
r

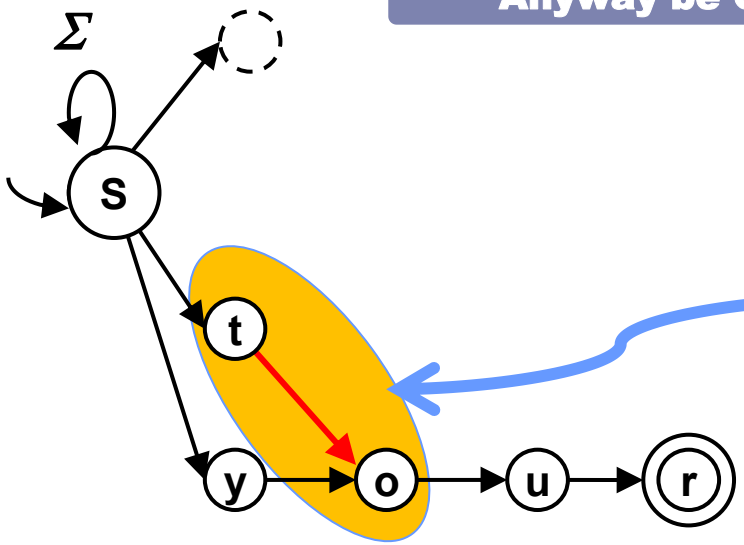
Search NFA for dictionary
D = {"add", "advanced", "algorithms", "to", "your", "algonqiuan", "adventures"}





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 an incorrect construction.
It would mistakenly add
word "tour" to the dictionary.

Effectivity

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

Transforming dictionary NFA of this shape to DFA does not increase 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, consequently we present only the transition table... :

Homework: Draw it!

	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...

.. 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...

.. continued

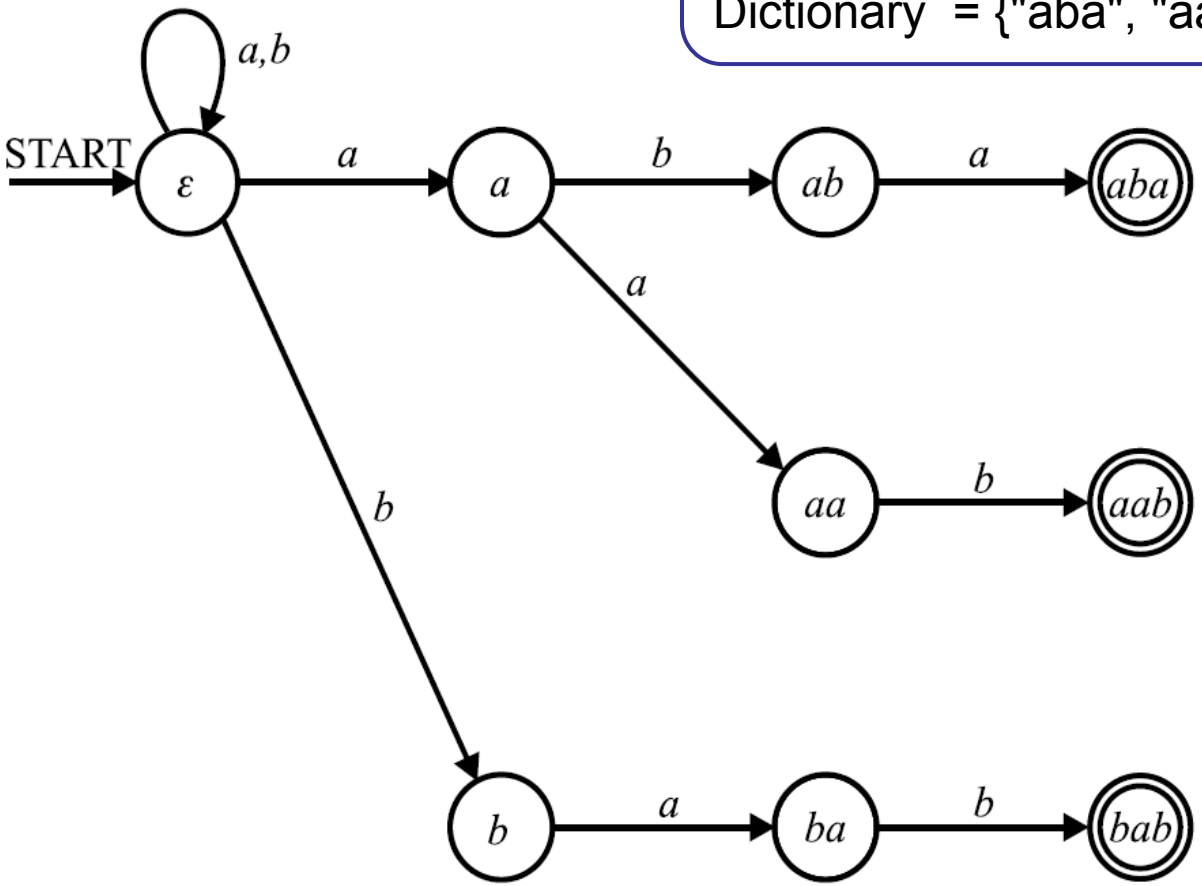
	a	c	d	e	g	h	i	l	m	n	o	q	r	s	t	u	v	y	
0,28	0,1	0	0	0	0	0	0,29	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,22, 32	0,1	0	0	0	0	0,23	0	0	0	0	0,33	0	0	0	0,32	0	0	0,34	
0,9	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	F
0,14	0,1	0	0	0,15	0	0	0	0	0	0	0	0	0	0	0,32	0	0	0,34	
0,23	0,1	0	0	0	0	0	0	0	0,24	0	0	0	0	0	0,32	0	0	0,34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

Example of dictionary automaton which transition diagram fits to one slide.

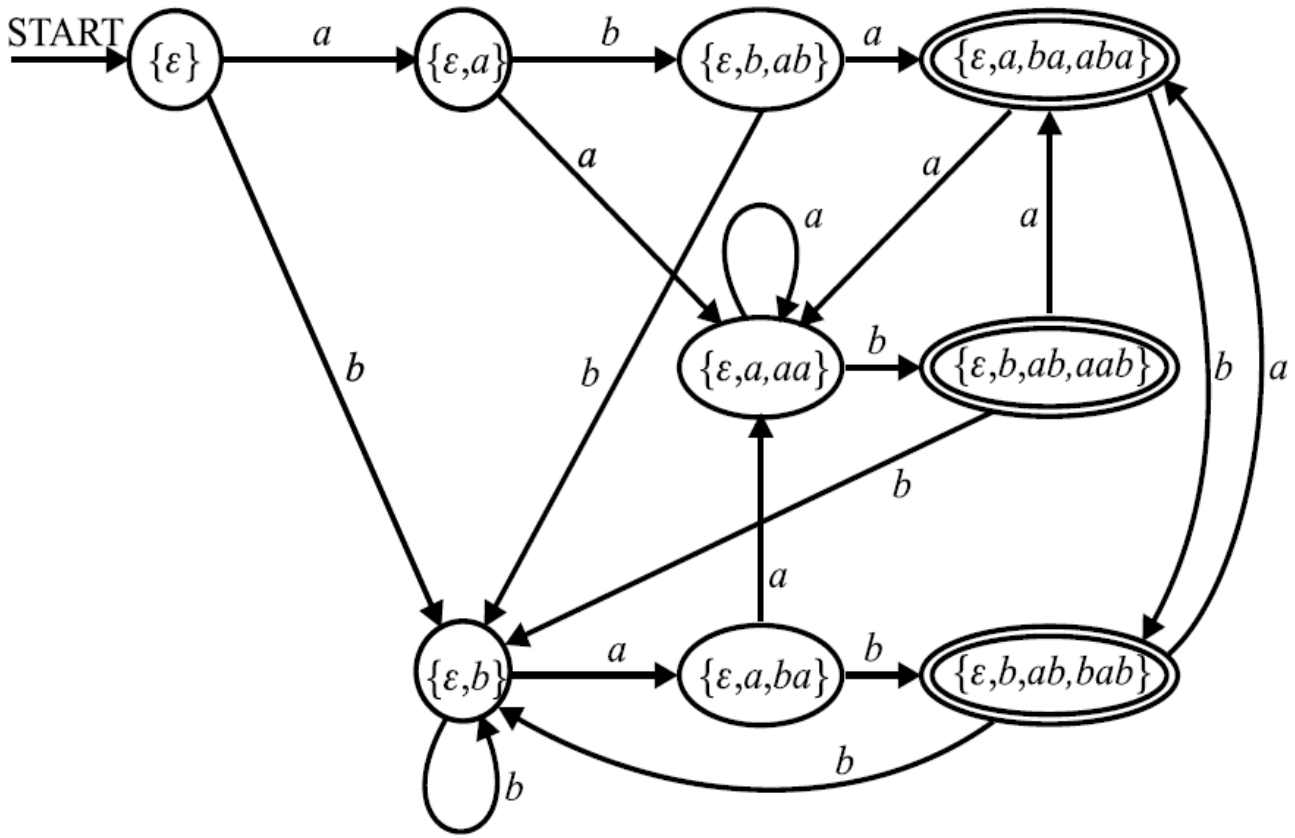
NFA

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



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], which 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].

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 match between text and 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

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], which 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 min of possible applications of operations of operations
   // delete, insert, rewrite to the strings shorter by one last char
   // 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. Element d[m][k] holds minimum Levenshtein distance of P from the substring t[x], t[x+1] ..., t[k],
where x ∈ {k-m+1-d[m][k], ..., k-m+1+d[m][k] }
and particular value of x is not known.

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 match between text and pattern

Text Search

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

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 of strings

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
Dist(A[1..n –1], B[1..m]),	...	**Insert**(A, n –1, B[m]) or **Delete**(B, m)
Dist(A[1..n], B[1..m –1]),	...	**Insert**(B, m –1, A[n]) or **Delete**(A, n)
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

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

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

Text search using Finite Automata brings in many possibilities regarding what can beeffectively found in many cases :

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

B. Elements of any language specified by DFA or NFA.
(Just add the loop at the start state labeled by whole alphabet.)

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

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

E. A string which is described by a regular expression.
(Regular expression can represent any regular language,
thus we can search for elements of any regular language)

F. Any subsequence of strings in A. - E..

G. Any union, intersection, ioncatenation, iteration of any of cases A. - G.