3.5. Three application of exact set matching

2015. 04. 14

HYE WON PARK

Exact Matching With Wild Card

- **Exact matching with wild card**
  - We modify the exact matching problem by introducing a character Φ
    - called a ***wild card***
    - matches **any single character**
  - Given a pattern P containing wild card, we want to find all occurrences of P in a text T
    - For example, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
P														

Exact Matching With Wild Card

- **Exact matching with wild card**
  - We modify the exact matching problem by introducing a character *Φ*
    - called a ***wild card***
    - matches **any single character**
  - Given a pattern *P* containing wild card, we want to find all occurrences of *P* in a text *T*
    - For example, *P* = *abΦΦcΦ* and *T* = *xabvccbababcax*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*
P		*a*	*b*	*Φ*	*Φ*	*c*	*Φ*							

Exact Matching With Wild Card

- **Exact matching with wild card**
  - We modify the exact matching problem by introducing a character Φ
    - called a ***wild card***
    - matches **any single character**
  - Given a pattern P containing wild card, we want to find all occurrences of P in a text T
    - For example, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
P								a	b	Φ	Φ	c	Φ	

Exact Matching With Wild Card

- **Exact matching with wild card**

  - If the number of permitted wild cards is **unbounded**, it is not known if the problem can be solved in linear time.

  - However, if the number of wild cards is **bounded** by a fixed constant (independent of the size of *P*) then the problem can be solved.
    - based on exact set pattern matching
    - runs in linear time

Exact Matching With Wild Card

- **Exact matching with wild card**

1. Let C be a vector of length $|T|$ initialized to all zero

2. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be the set of maximal substrings of P that do not contain any wild card. Let $l_1, l_2, \ldots, l_k$ be the starting positions in P of each of these substrings.
   - For example (1), if $P = \boldsymbol{ab\Phi\Phi c\Phi}$, then $\mathcal{P} = \{ ab, c \}$ and $l_1 = 1$, $l_2 = 5$
   - For example (2), if $P = \boldsymbol{ab\Phi\Phi c\Phi ab\Phi\Phi}$, then $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

Exact Matching With Wild Card

- **Exact matching with wild card**

3. Using the Aho-Corasick algorithm, find for each string P_i in $\mathcal{P}$, all starting positions of P_i in text T. For each starting location j of P_i in T, increment the count in cell $j - l_i + 1$ of C by one.
   - For example, if the second copy of string *ab* is found in T starting at position 18, then cell 12 of C is incremented by one

4. Scan vector for any cell with value k. there is an occurrence of P in T staring at position if and only if $C(p) = k$.

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = abΦΦcΦ$ and $T = xabvccbababcax$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- $P = abΦΦcΦ$
  - $\mathcal{P} = \{ ab, c \}$ and $l_1 = 1$, $l_2 = 5$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c \}$ and $l_1 = 1$, $l_2 = 5$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	0	1	0	0	0	0	0	0	0	0	0	0	0	0

  - $P_1 \rightarrow 2$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\, ab, c \,\}$ and $l_1 = 1$, $l_2 = 5$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*
C	0	1	0	0	0	0	0	1	0	0	0	0	0	0

  - $P_1 \rightarrow 2, 8$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\ ab, c\ \}$ and $l_1 = 1$, $l_2 = 5$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0

  - $P_1 \longrightarrow 2, 8, 10$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\ ab, c\ \}$ and $l_1 = 1$, $l_2 = 5$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C	1 ←	0 ←	0 ←	0 ←	0	0	0	0	0	0	0	0	0	0	*c*

start(l_2)

  - $P_1 \rightarrow 2, 8, 10$
  - $P_2 \rightarrow 5$ ($j - l_i + 1$ ➡ 5-5+1=1)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\ ab, c\ \}$ and $l_1 = 1$, $l_2 = 5$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C	1	1 ←	0 ←	0 ←	0 ←	0	0	0	0	0	0	0	0	0	*c*

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6$ ($j - l_i + 1$ ➡ 6-5+1=2)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c \}$ and $l_1 = 1$, $l_2 = 5$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	ab
C	1	1	0	0	0	0	0	1 ←	0 ←	0 ←	0 ←	0	0	0	c

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6, 12$ ($j - l_i + 1$ ➡ 12-5+1=8)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\ ab, c\ \}$ and $l_1 = 1,\ l_2 = 5$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C	1	1	0	0	0	0	0	1	0	0	0	0	0	0	*c*

Add

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	1	2	0	0	0	0	0	2	0	1	0	0	0	0	*ab + c*

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c \}$ and $l_1 = 1$, $l_2 = 5$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	1	$\boxed{2}$	0	0	0	0	0	$\boxed{2}$	0	1	0	0	0	0

  - $C(p) = k$
    - $C(2) = 2$ and $C(8) = 2$
    - There is an occurrence of P in T starting at position 2 and 8

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example1, $P = ab\Phi\Phi c\Phi$ and $T = xabvccbababcax$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
		a	b	Φ	Φ	c	Φ							
								a	b	Φ	Φ	c	Φ	

  - $P = ab\Phi\Phi c\Phi$
    - There is an occurrence of P in T starting at position 2 and 8

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = ab\Phi\Phi c\Phi ab\Phi\Phi$ and $T = xabvccbababcax$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- $P = ab\Phi\Phi c\Phi ab\Phi\Phi$
  - $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$
    $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	0	1	0	0	0	0	0	0	0	0	0	0	0	0

  - $P_1 \rightarrow 2$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$
    $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	0	1	0	0	0	0	0	1	0	0	0	0	0	0

  - $P_1 \rightarrow 2, 8$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = ab\Phi\Phi c\Phi ab\Phi\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0

  - $P_1 \rightarrow 2, 8, 10$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	
T	*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C	1	0	0	0	0	0	0	0	0	0	0	0	0	0	*c*

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5(j - l_i + 1 \Rightarrow 5\text{-}5+1=1)$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

    $\mathcal{P}$ ={ ab, c, ab } and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	ab
C	1	1 ←	0 ←	0 ←	0 ←	0	0	0	0	0	0	0	0	0	c

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6$ ($j - l_i + 1$ ➡ 6-5+1=2)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\ ab, c, ab\ \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	
T	*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*	
C	0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C	1	1	0	0	0	0	0	1	0	0	0	0	0	0	*c*

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6, 12$ ($j - l_i + 1$ ➡ 6-5+1=2)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = ab\Phi\Phi c\Phi ab\Phi\Phi$ and $T = xabvccbababcax$

    $\mathcal{P} = \{\ ab,\ c,\ ab\ \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	-4	…	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T			*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*	
C			0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C			1	1	0	0	0	0	0	1	0	0	0	0	0	0	*c*
C	1 ←	… ←	0 ←	0	0	0	0	0	0	0	0	0	0	0	0	0	*ab*

↑ start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6, 12$
- $P_3 \rightarrow 2$ ($j - l_i + 1$ ➡ $2-7+1 = -4$)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1, l_2 = 5, l_3 = 7$

	-4	…	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T			*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*	
C			0	1	0	0	0	0	0	1	0	1	0	0	0	0	*ab*
C			1	1	0	0	0	0	0	1	0	0	0	0	0	0	*c*
C	1	…	0	1	0	0	0	0	0	0	0	0	0	0	0	0	*ab*

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6, 12$
- $P_3 \rightarrow 2, 8$ ($j - l_i + 1$ ➡ 8-7+1 = 2)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = ab\Phi\Phi c\Phi ab\Phi\Phi$ and $T = xabvccbababcax$

    $\mathcal{P}$ ={ ab, c, ab } and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	-4	…	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T			x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C			0	1	0	0	0	0	0	1	0	1	0	0	0	0	ab
C			1	1	0	0	0	0	0	1	0	0	0	0	0	0	c
C	1	…	0	1	0	1	0	0	0	0	0	0	0	0	0	0	ab

start(l_2)

- $P_1 \rightarrow 2, 8, 10$
- $P_2 \rightarrow 5, 6, 12$
- $P_3 \rightarrow 2, 8, 10$ ($j - l_i + 1$ ➡ 10-7+1 = 4)

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

$\mathcal{P}$ ={ ab, c, ab } and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	-4	…	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T			x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C			0	1	0	0	0	0	0	1	0	1	0	0	0	0	ab
C			1	1	0	0	0	0	0	1	0	0	0	0	0	0	c
C	1	…	0	1	0	1	0	0	0	0	0	0	0	0	0	0	ab

Add

	-4	…	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T			x	a	b	v	c	c	b	a	b	a	b	c	a	x	
C	1	…	1	3	0	0	0	0	0	2	0	1	0	0	0	0	$ab + c$

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

    $\mathcal{P} = \{ ab, c, ab \}$ and $l_1 = 1$, $l_2 = 5$, $l_3 = 7$

	-4	**…**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T			*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*
C	1	…	1	$\boxed{3}$	0	0	0	0	0	2	0	1	0	0	0	0

  - $C(p) = k$
    - $C(2) = 3$
    - There is an occurrence of *P* in *T* starting at position 2

Exact Matching With Wild Card

- **Exact matching with wild card**
  - For example2, $P = abΦΦcΦabΦΦ$ and $T = xabvccbababcax$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	*x*	*a*	*b*	*v*	*c*	*c*	*b*	*a*	*b*	*a*	*b*	*c*	*a*	*x*
		a	*b*	*Φ*	*Φ*	*c*	*Φ*	*a*	*b*	*Φ*	*Φ*			

  - $P = abΦΦcΦabΦΦ$
  - There is an occurrence of P in T starting at position 2

Exact Matching With Wild Card

- **Complexity**

  - The time used by the Aho-Corasick algorithm to build the keyword for $\mathcal{P}$ is $O(m)$

  - The time to search for occurrences in T of patterns from $\mathcal{P}$ is $O(n + z)$
    - Where $|T| = n$ and z is the number of occurrences

  - As a result, total time complexity is $O(m+ n + z)$.

Exact Matching With Wild Card

- **Complexity**

  - Whenever an occurrence of a pattern from $\mathcal{P}$ is found in T, exactly one cell in C is incremented
    - Furthermore, a cell can be incremented to at most k

  - Search time complexity is $O(n + z)$ and z must be bounded by kn, and the algorithm runs in $O(n + kn)$ time.

  - Then $kn \geq n$ ($k \geq 1$), *time complexity is* $\mathrm{O}(kn)$.

  - As a result, total time complexity is $O(m + \mathrm{k}n)$.

Exact Matching With Wild Card

- **Complexity**
  - If the number of wild cards in a pattern P is bounded by a constant,
  - k become constant value. Then the exact matching problem with wild cards in the Pattern can be solved in $O(m + n)$ time.

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - Suppose we have a rectangular digitized picture *T*, where each point is given a number indication its color and brightness
  - We are also given a smaller rectangular picture *P*, which also is digitized
  - We want to find all occurrences (possibly overlapping) of the smaller picture in the larger one

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - Application of two-dimensional exact matching are hard to find
  - Two-dimensional matching that is inexact, allowing some errors, is a more realistic problem
  - Its solution requires more complex techniques of the type

Two-dimensional Exact Matching

- **Two-dimensional exact matching**

  - Let

    n be the total number of points in T

    m be the number of points in P

    m' be the number of rows in P

  - Just as in exact string matching, we want to find the smaller picture in the larger one in $O(n + m)$ time, where $O(nm)$ is the time for the obvious approach

  - Assume for now that each of the rows of P are distinct
    - Later we will relax this assumption

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - ***O*(*nm*)**
    - **ex) m=9, n=100**

P (3x3) =

a	*s*	*c*
q	*w*	*s*
d	*a*	*a*

T (10x10) =

c	*c*	*c*	*k*	…	*q*
e	*b*	*s*	*h*		*w*
c	*d*	*a*	*p*		*e*
d	*e*	*w*	*r*		*t*
…					*u*
a	*a*	*b*	*b*	*w*	*i*

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - ***O*(*nm*)**
    - **ex) m=9, n=100**

P = (3x3)

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

"Compare m time "

T = (10x10)

c	*c*	*c*	*k*	…	*q*
e	*b*	*s*	*h*		*w*
c	*d*	*a*	*p*		*e*
d	*e*	*w*	*r*		*t*
…					*u*
a	*a*	*b*	*b*	*w*	*i*

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - $O(nm)$
    - **ex) m=9, n=100**

P (3x3) =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

"Compare m time "

T (10x10) =

c	*c*	*c*	*k*	…	*q*
e	*b*	*s*	*h*		*w*
c	*d*	*a*	*p*		*e*
d	*e*	*w*	*r*		*t*
…					*u*
a	*a*	*b*	*b*	*w*	*i*

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - $O(nm)$
    - **ex) m=9, n=100**

P = (3x3)

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

"Compare m time "

T = (10x10)

c	*c*	*c*	*k*	…	*q*
e	*b*	*s*	*h*		*w*
c	*d*	*a*	*p*		*e*
d	*e*	*w*	*r*		*t*
…					*u*
a	*a*	*b*	*b*	*w*	*i*

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - ***O*(*nm*)**
    - **ex) m=9, n=100**

"Compare m time "

T (10x10) =

c	*c*	*c*	*k*	…	*q*
e	*b*	*s*	*h*		*w*
c	*d*	*a*	*p*		*e*
d	*e*	*w*	*r*		*t*
…					*u*
a	*a*	*b*	*b*	*w*	*i*

P (3x3) =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

If match is worst case, Time complexity is O(*nm*)

Two-dimensional Exact Matching

- **Two-dimensional exact matching**
  - The method is divided into two phases.
  - In the **first phase**, **search for all occurrences** of each of the row of ***P*** among the rows of ***T***
  - In the **second phase**, **scan each column of M**, looking for an occurrence of the string 1,2,…,n' in consecutive cells in a single column

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*

$\mathcal{P}$ = { *aab, ack, bac* }

- Add an end of **row marker** (**some character** not in the alphabet) to each row of T

ex) *, &, $, #

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$

$\mathcal{P}$ = { *aab, ack, bac* }

- Add an end of **row marker** (**some character** not in the alphabet) to each row of T
  ex) *, &, $, #

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*

$\mathcal{P}$ = { *aab, ack, bac* }

- Add an end of **row marker** (**some character** not in the alphabet) to each row of T **ex) *, &, $, #**

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$

$\mathcal{P}$ = { *aab, ack, bac* }

- Add an end of **row marker** (**some character** not in the alphabet) to each row of T
  ex) *, &, $, #

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*

$\mathcal{P}$ = { *aab, ack, bac* }

- Add an end of **row marker** (**some character** not in the alphabet) to each row of T
  ex) *, &, $, #

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*	$

$\mathcal{P}$ = { *aab, ack, bac* }

- Add an end of **row marker** (**some character** not in the alphabet) to each row of T
  ex) *, &, $, #

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*	$

…

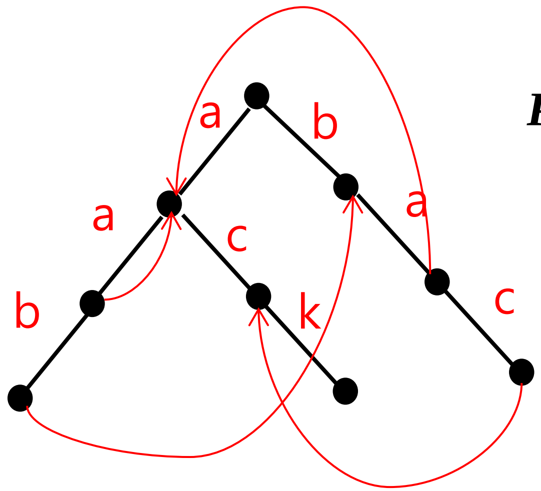
length *n*

$\mathcal{P}$ = { *aab, ack, bac* }

- Concatenate these rows together to form a single text string T' of length $O(n)$

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**



P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

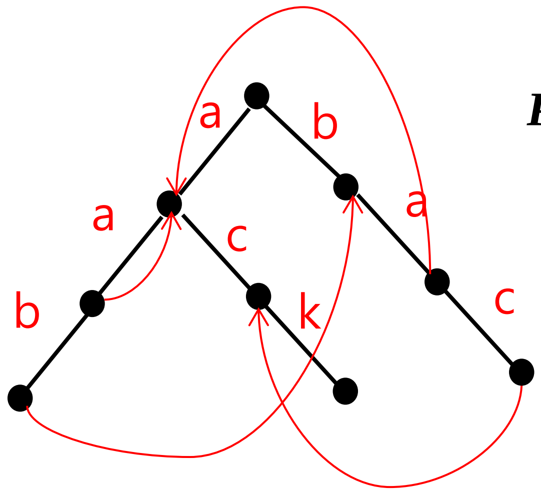
a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*	$	…

$\mathcal{P}$ = { *aab, ack, bac* }

- Then, treating each row of *P* as a separate of *P*, use the **Aho-Corasick algorithm** to search for all occurrences in *T′* of any row of *P*

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**



P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*	$

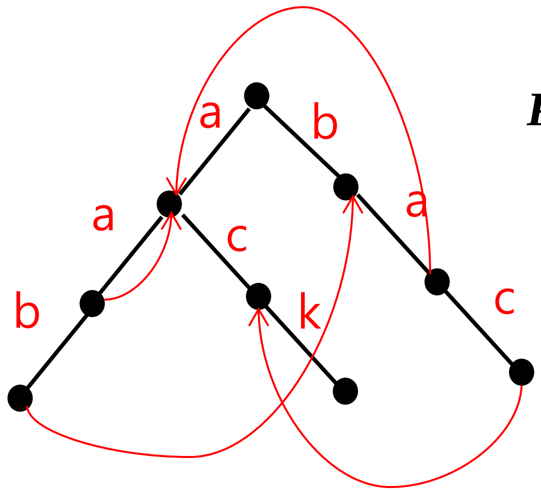
…

$\mathcal{P}$ = { ***aab, ack, bac*** }

- Since *P* is rectangular, all rows have the same width.
- So no row is a proper substring of another and we can use the simpler version of Aho-Corasick.

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**



P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

T' =

a	*a*	*b*	*b*	*c*	*f*	**$**	*a*	*c*	*k*	*d*	*c*	*e*	**$**	*b*	*a*	*c*	*a*	*a*	*b*	**$**	…

$\mathcal{P}$ = { ***aab, ack, bac*** }

- Hence the first phase identifies all occurrences of complete rows of *P* in complete rows of *T* and take **O(*n*+*m*)** time.

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*	$

…

$\mathcal{P}$ = { *aab, ack, bac* }

M =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

- ***M*** with the same dimensions as ***T*** is another array.

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

T' =

a	*a*	*b*	*b*	*c*	*f*	$	*a*	*c*	*k*	*d*	*c*	*e*	$	*b*	*a*	*c*	*a*	*a*	*b*	$

$\mathcal{P}$ = { ***aab, ack, bac*** }

ex) i = 1, 2, 3

M =

1					
2					
3			*1*		
			2		
			3		

- Whenever an occurrence of row *i* of *P* is found starting at position(*p*, *q*) of *T*, write the number ***i*** in position (*p*, *q*) of array *M*

Two-dimensional Exact Matching

- **Two-dimensional exact matching(First phase of method)**

M =

1					
2					
3			*1*		
			2		
			3		

***P* occurs in *T* when its upper left corner is at position (1, 1) and (3, 4)**

P =

a	*a*	*b*
a	*c*	*k*
b	*a*	*c*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
b	*a*	*c*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*b*	*a*	*c*

Two-dimensional Exact Matching

- **Two-dimensional exact matching(Second phase of method)**

    - In the second phase, scan each column of M, looking for an occurrence of the string 1, 2, . . . , n' in consecutive cells in a single column

    - Phase two can be implemented in $O(n' + m) = O(n + m)$ time by applying any linear-time exact matching algorithm to each column of M

Two-dimensional Exact Matching

- **Two-dimensional exact matching**

  - Now suppose that the rows of *P* are not all distinct

  - Then first find all identical rows and give them a common label
    - this is easily done during the construction of the keyword tree for the row patterns

Two-dimensional Exact Matching

- **Two-dimensional exact matching**

P =

a	*a*	*b*
a	*c*	*k*
a	*a*	*b*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
a	*a*	*b*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*a*	*a*	*b*

T' =

a	*a*	*b*	*b*	*c*	$	*a*	*c*	*k*	*d*	*c*	$	*a*	*a*	*b*	*a*	*a*

$\mathcal{P}$ = { aab, ack }

Two-dimensional Exact Matching

- **Two-dimensional exact matching**

T' =																	
	a	***a***	***b***	***b***	***c***	**$**	***a***	***c***	***k***	***d***	***c***	**$**	***a***	***a***	***b***	***a***	***a***

$\mathcal{P}$ **= { aab, ack }**

M =

1					
2					
1			***1***		
			2		
			1		

Two-dimensional Exact Matching

- **Two-dimensional exact matching**

M =

1					
2					
1			*1*		
			2		
			1		

_P_ occurs in _T_ when its upper left corner is at position (1,1), (3,4)

P =

a	*a*	*b*
a	*c*	*k*
a	*a*	*b*

T =

a	*a*	*b*	*b*	*c*	*f*
a	*c*	*k*	*d*	*c*	*e*
a	*a*	*b*	*a*	*a*	*b*
a	*v*	*s*	*a*	*c*	*k*
d	*d*	*a*	*a*	*a*	*b*

Two-dimensional Exact Matching

- **Theorem 3.5.2**

  If *T* and *P* are rectangular pictures with m and n cells, respectively, then all exact occurrences of *P* in *T* can be found in $O(n + m)$ time, improving upon the naïve method, which takes $O(nm)$ time.

Regular expression

- **Regular expression**

  Way to specify a set of related strings.

  ex) ax, ay, az, bx, by, bz, cx, cy, cz

  ➔ [a b c] – [x y z]

- **Regular expression pattern matching**

  examine the problem of finding substrings of a text string

  that match one of the strings specified by a given regular expression

Failure functions

ex) R = [ED]-[EN]-L-[SAN]-*x*-*x*-[DE]-*x*-E-L

T = … A G E N L S S E D E E L E B …

Failure functions

ex) R = [ED]-[EN]-L-[SAN]-*x*-*x*-[DE]-*x*-E-L

[**E**D]-[E**N**]-**L**-[**S**AN]-***x***-***x***-[**D**E]-***x***-**E**-**L**

T = … A G <u>E N L S S E D E E L</u> E B …

Failure functions

ex) R = [ED]-[EN]-L-[SAN]-*x*-*x*-[DE]-*x*-E-L

[ED]-[EN]-L-[SAN]-*x*-*x*-[DE]-*x*-E-L

D	E	L	S	A	A	D	A	E	L
D	N	L	A	A	A	D	A	E	L
D	N	L	N	A	A	E	A	E	L
E	E	L	S	A	A	D	A	E	L
E	N	L	A	A	A	D	A	E	L
E	N	L	N	A	A	E	A	E	L

⋮

Failure functions

ex) R = [ED]-[EN]-L-[SAN]-*x*-*x*-[DE]-*x*-E-L

[ED]-[EN]-L-[SAN]-*x*-*x*-[DE]-*x*-E-L

2 2 3 20 20 2 20

×

192000 number of case

Formal definitions

- **Formal definition of a regular expression**
  - Σ
    - A single character from Σ is a regular expression
  - ε
    - The symbol ε is a regular expression
      (represents the **empty string**)
  - *RR*
    - A regular expression followed by another regular expression is a regular expression

Formal definitions

- **Formal definition of a regular expression**
  - $R + R$
    - Two regular expressions separated by the symbol “+” form a regular expression
  - (R)
    - A regular expression enclosed in parentheses is a regular expression
  - $(R)^*$
    - A regular expression enclosed in parentheses and followed by the symbol “*” is a regular expression
    - The symbol * is called the Kleene closure

      (R can be repeated any number of times)

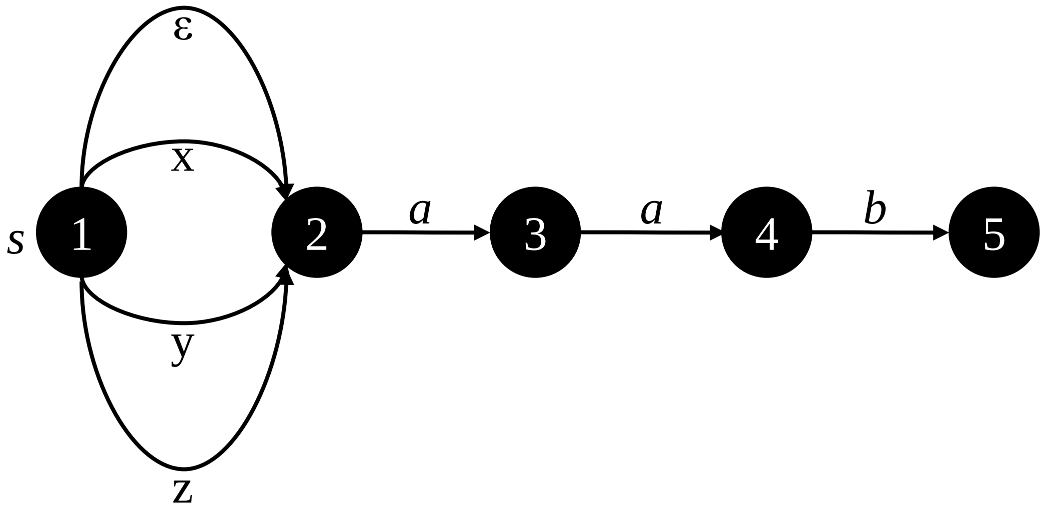
Formal definitions

- +

$$\left.\begin{array}{r} a\ a\ b \\ x\ a\ a\ b \\ y\ a\ a\ b \\ z\ a\ a\ b \end{array}\right] \Rightarrow (\varepsilon + x + y + z)\ a\ a\ b$$

Formal definitions

- $s = (\varepsilon + x + y + z)\, a\, a\, b$



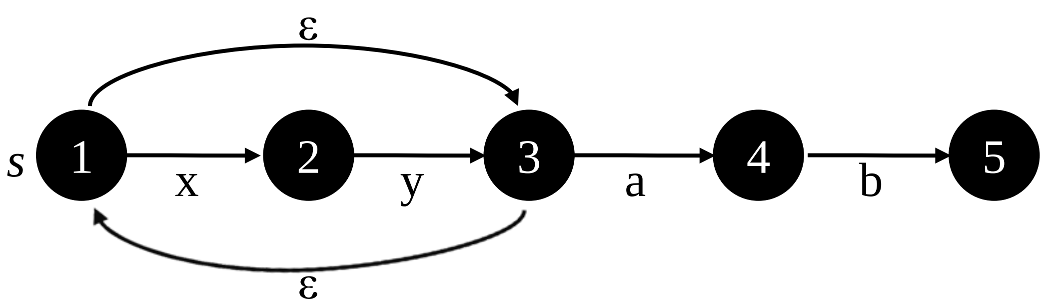
Formal definitions

- *

$$\left.\begin{array}{r} a\ b \\ \underline{x\ y}\ a\ b \\ \underline{x\ y}\ \underline{x\ y}\ a\ b \\ \underline{x\ y}\ \underline{x\ y}\ \underline{x\ y}\ a\ b \\ \vdots \end{array}\right\} \Rightarrow (xy)*a\ b$$

Formal definitions

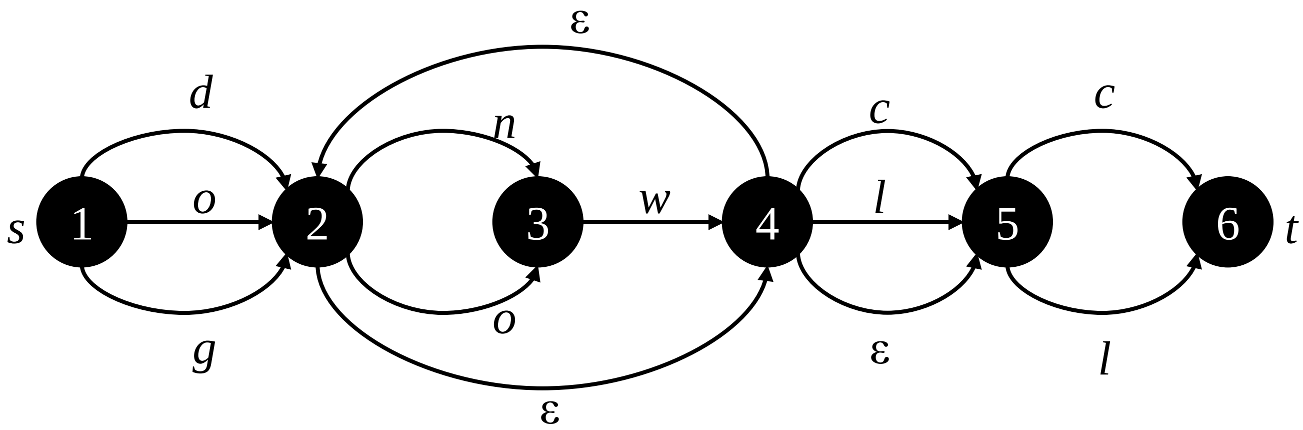
- $s = (xy)^* a b$



Failure functions

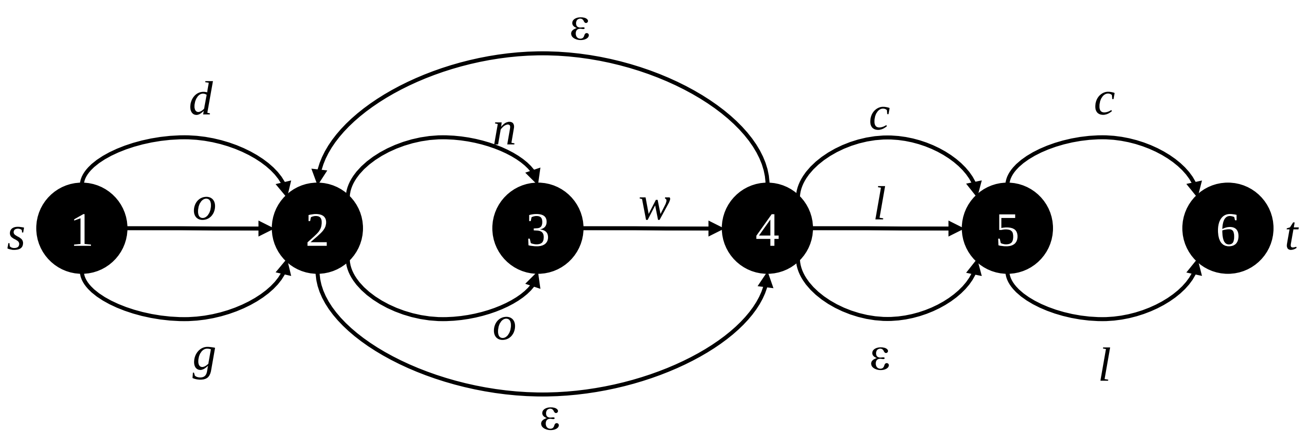
- **Directed graph**

$$R = (d+o+g)((n+o)w)^*(c+l+\varepsilon)(c+l)$$



Failure functions

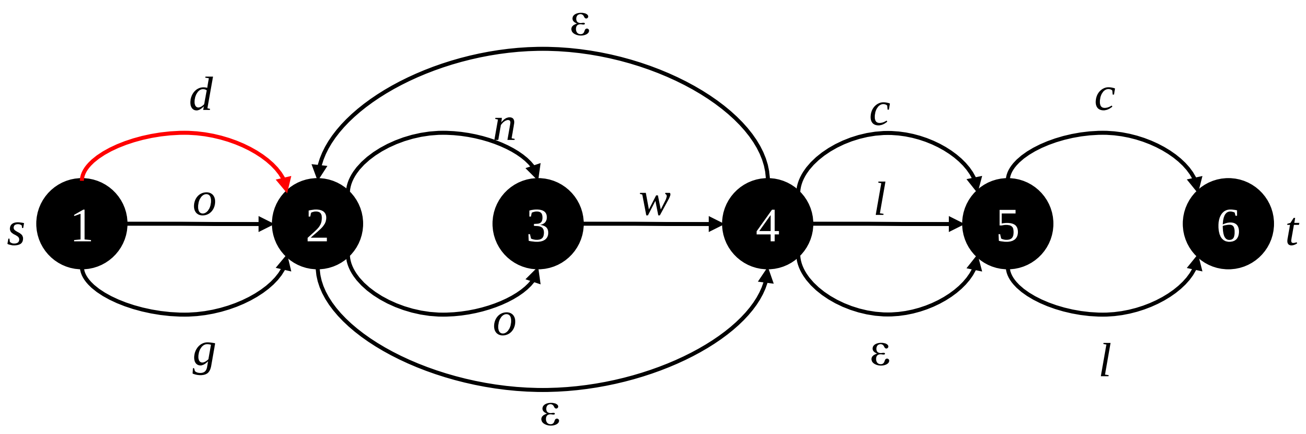
- **Directed graph**



$T = d\ n\ w\ o\ w\ c\ l$

Failure functions

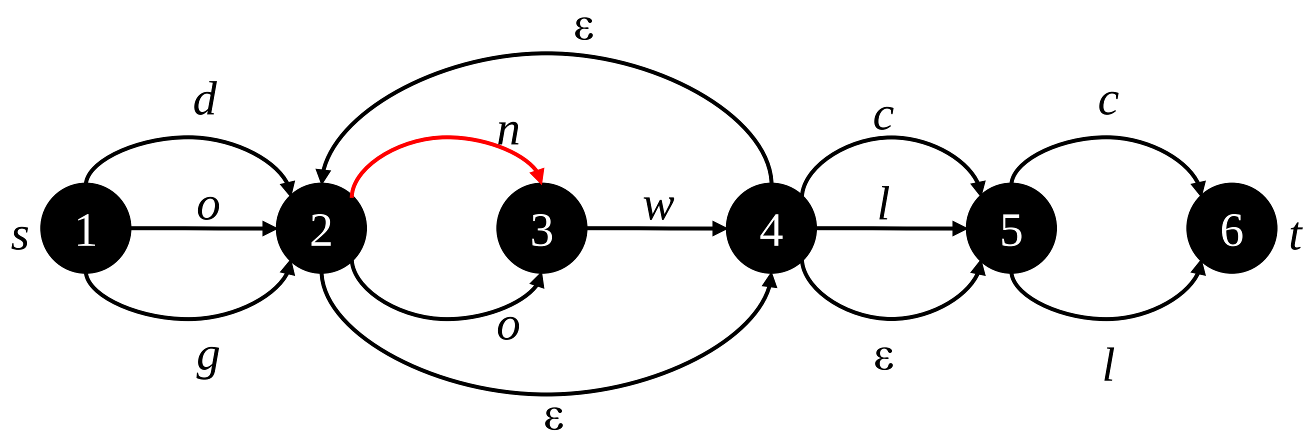
- **Directed graph**



$T = \textcolor{red}{\mathbf{d}}\ n\ w\ o\ w\ c\ l$

Failure functions

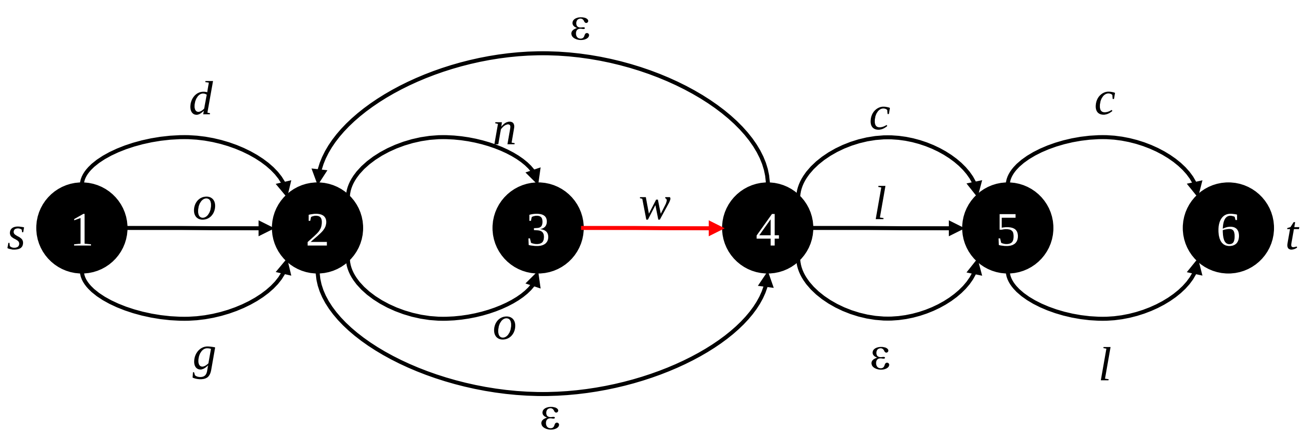
- **Directed graph**



$T = d\ \mathbf{n}\ w\ o\ w\ c\ l$

Failure functions

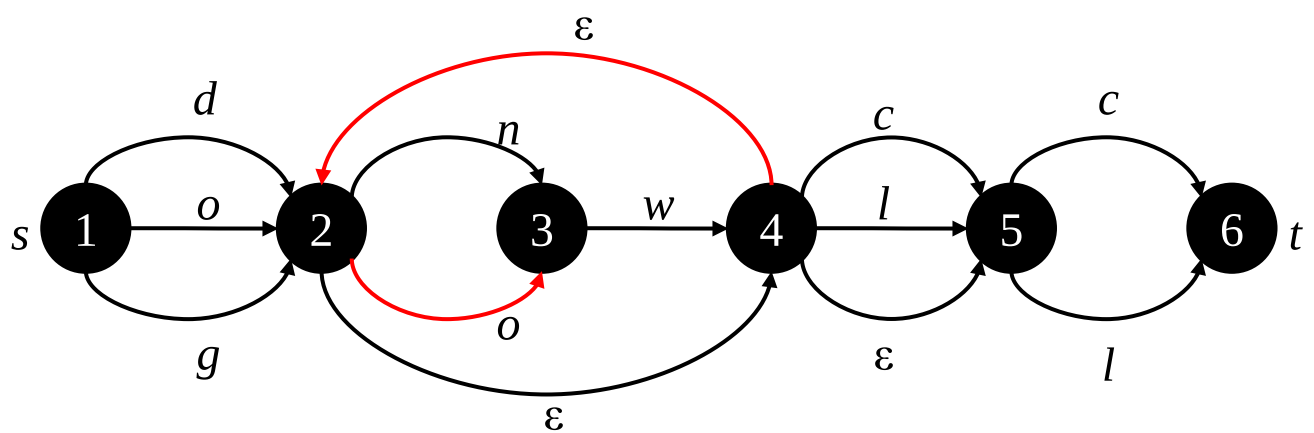
- **Directed graph**



$T = d\ n\ \mathbf{w}\ o\ w\ c\ l$

Failure functions

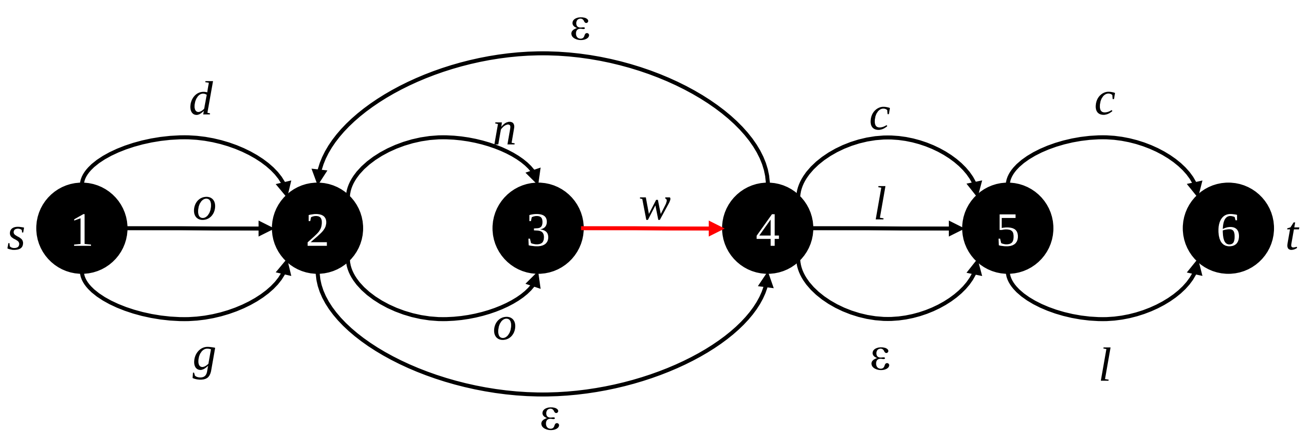
- **Directed graph**



$T = d\ n\ w\ \mathbf{o}\ w\ c\ l$

Failure functions

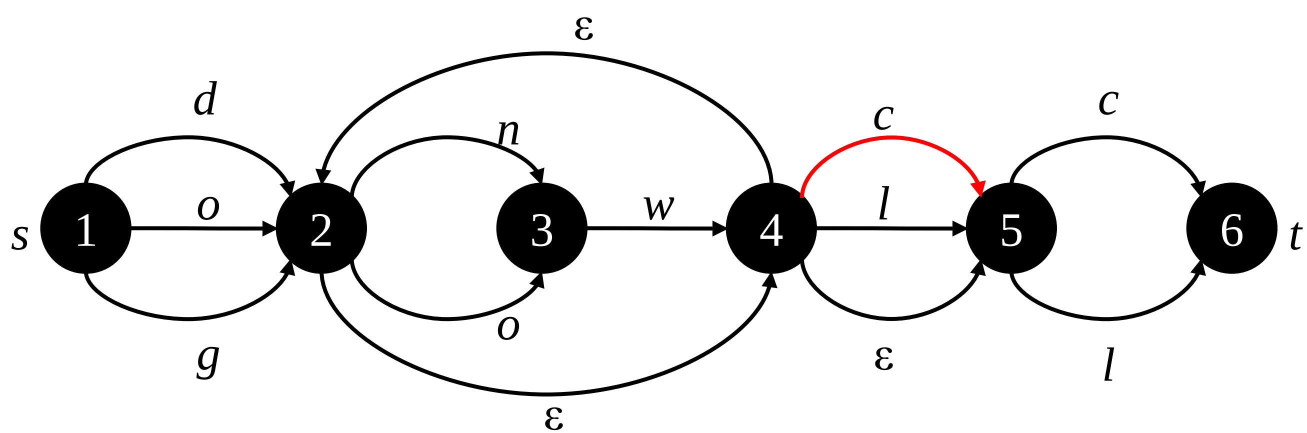
- **Directed graph**



$T = d\ n\ w\ o\ \mathbf{w}\ c\ l$

Failure functions

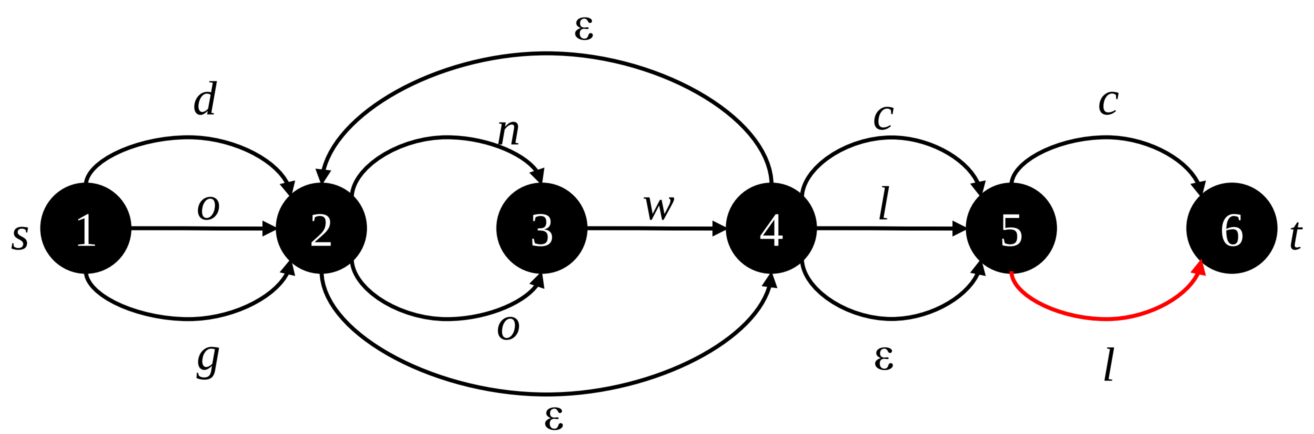
- **Directed graph**



$T = d\ n\ w\ o\ w\ \mathbf{c}\ l$

Failure functions

- **Directed graph**



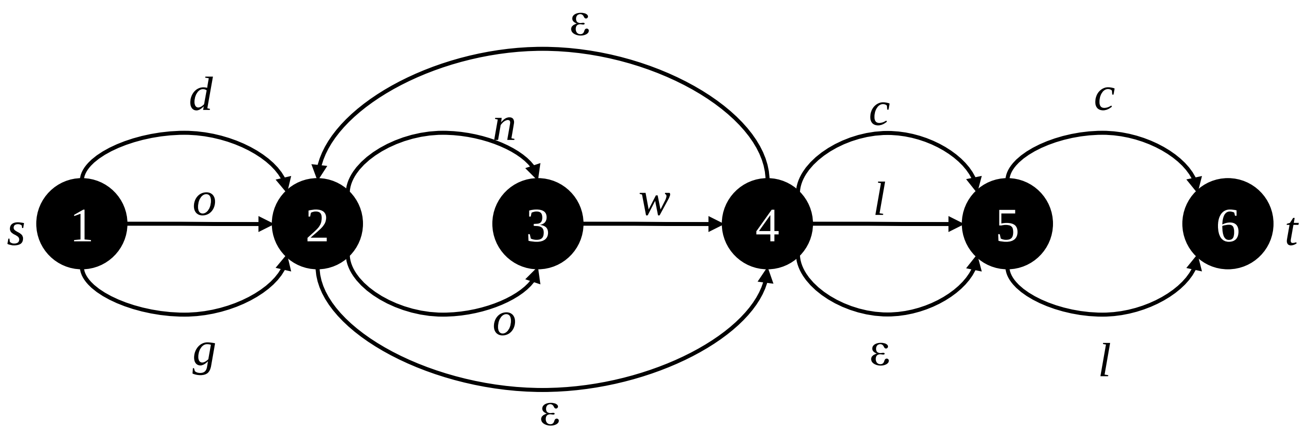
$T = d\ n\ w\ o\ w\ c\ \boldsymbol{l}$

Formal definitions

- **Searching for matches**
  - To sarch for a substring in *T* that matches the regular expression *R*, consider the simpler problem of determining whether some ***prefix of T* matches *R*.**
  - Let
    - *N(0)*
      - Set of nodes consisting of node s plus all nodes of G(R) that are reachable from node s by traversing edges labeled **ε**
    - node *v*
      - is in set ***N(i)*** , for *i* > 0, *v* can be reached from some node in ***N(i-1)*** by trabersing an edge labeled ***T(i)***
  - It is constructive **rule for finding set *N(i)*** from set ***N(i-1)*** and character ***T(i)***

Failure functions

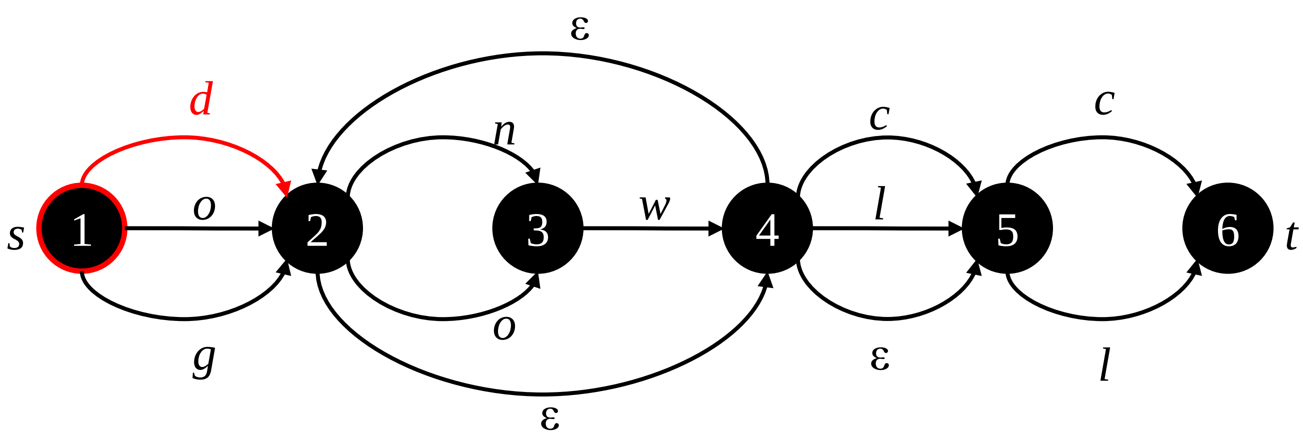
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1							

Failure functions

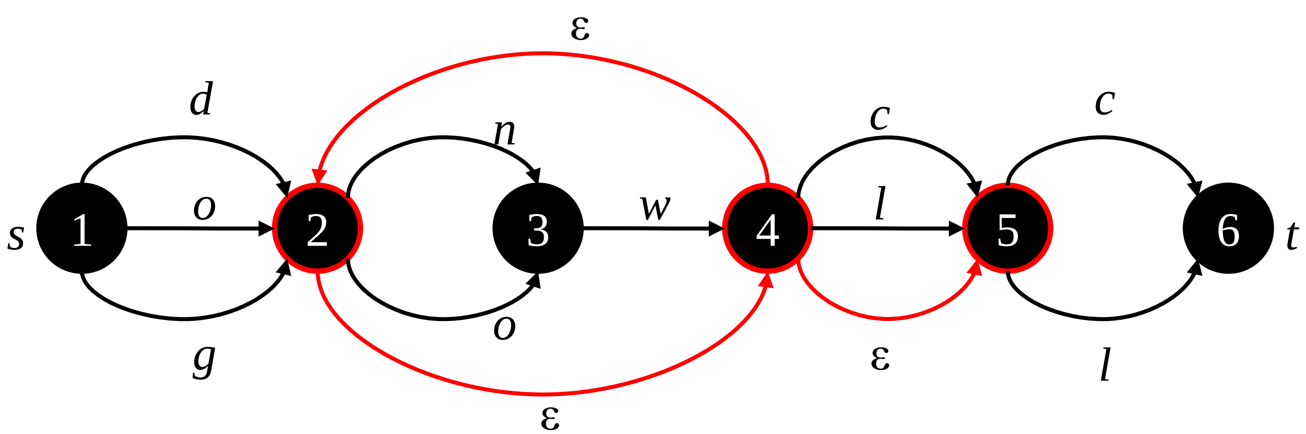
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1							

Failure functions

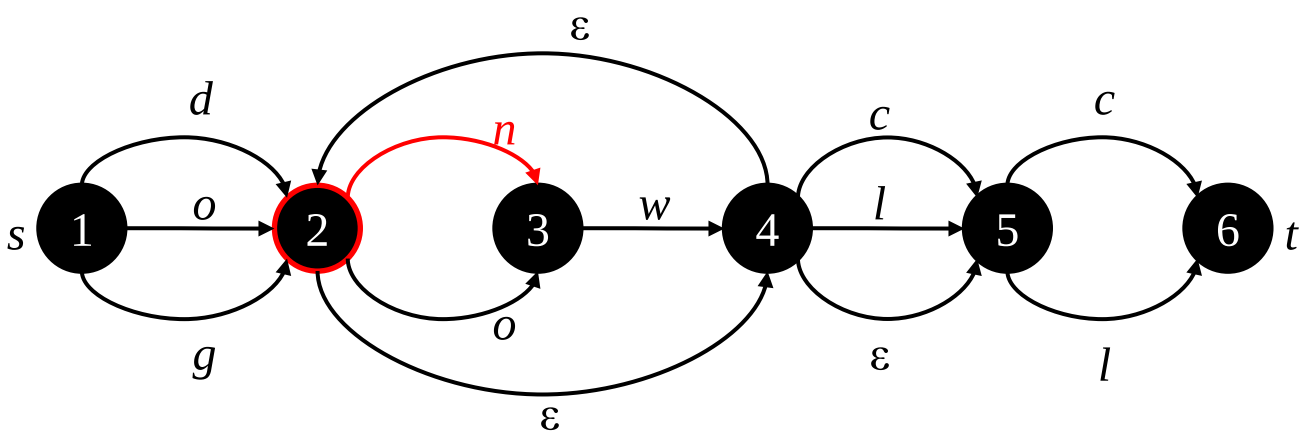
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5						

Failure functions

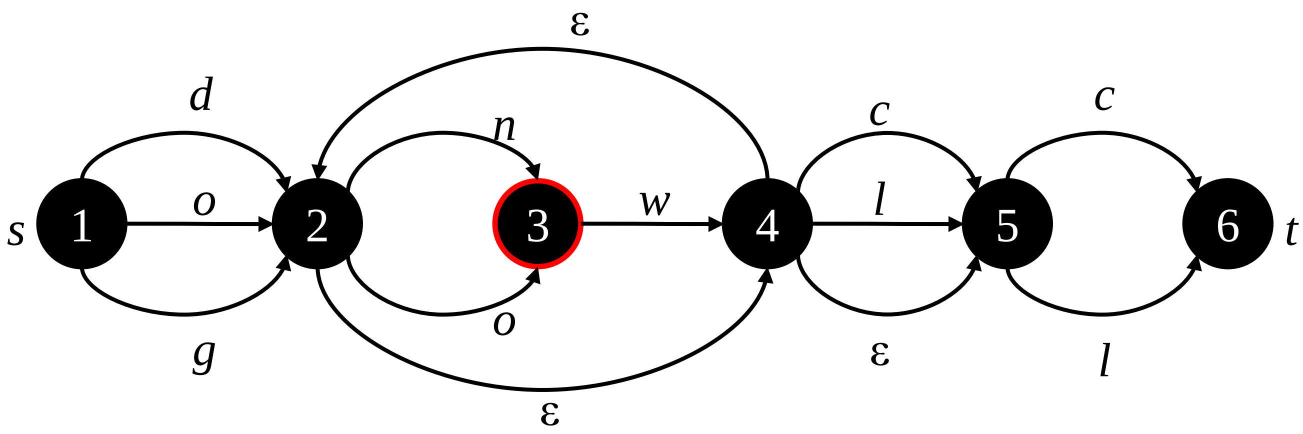
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5						

Failure functions

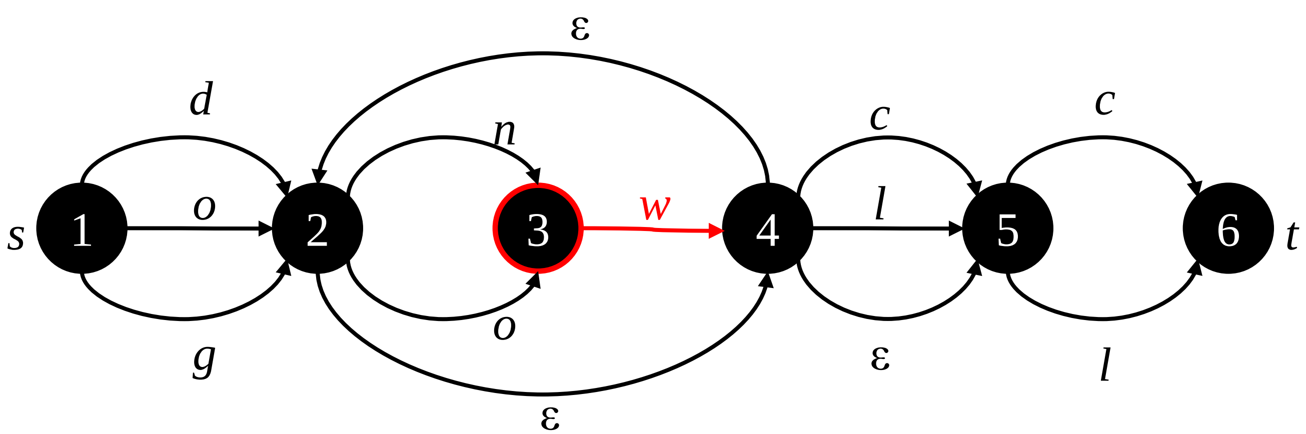
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3					

Failure functions

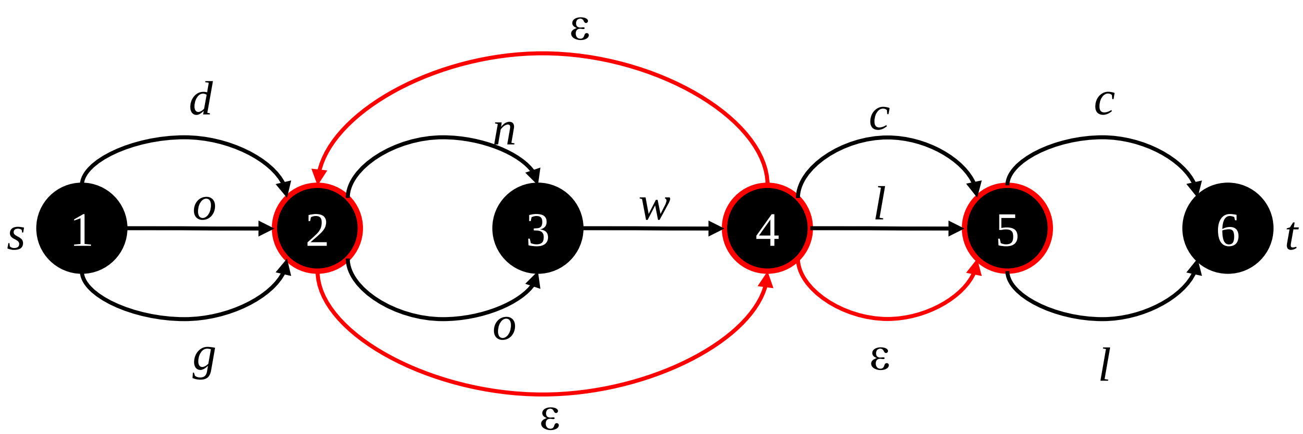
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3					

Failure functions

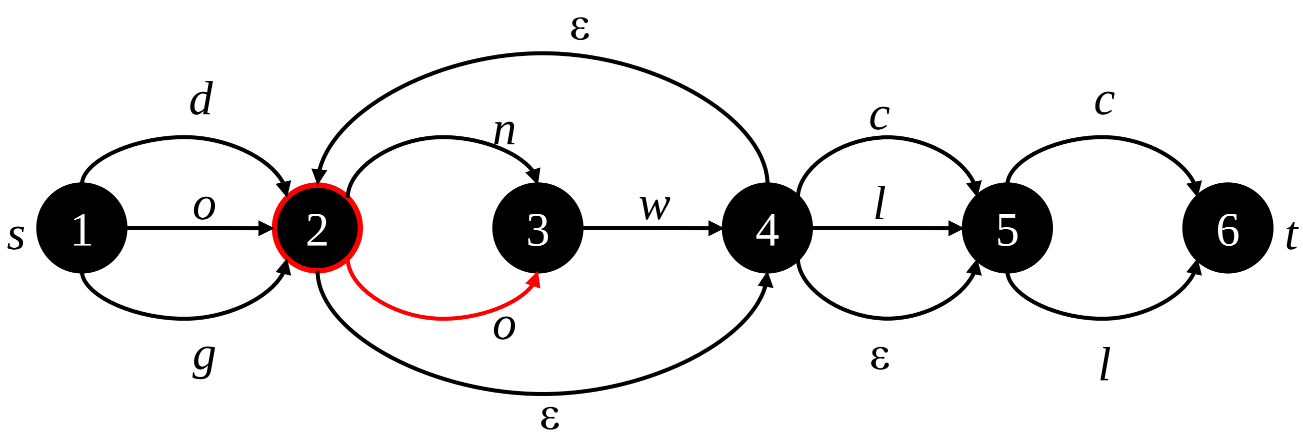
- **Directed graph**



i	0	1	2	3	4	5	6	7
T(*i*)		*d*	*n*	*w*	*o*	*w*	*c*	*l*
N(*i*)	1	2, 4, 5	3	2, 4, 5				

Failure functions

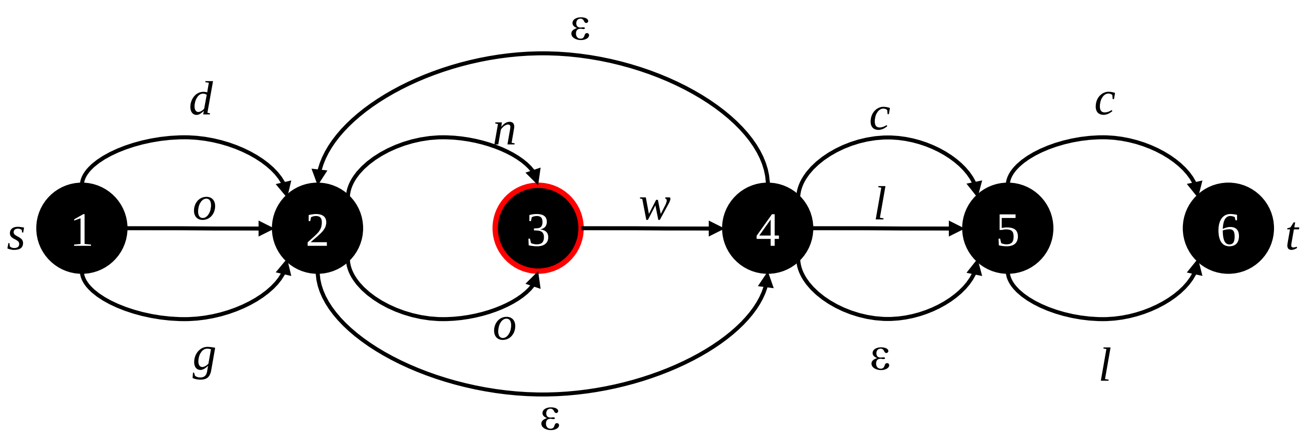
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5				

Failure functions

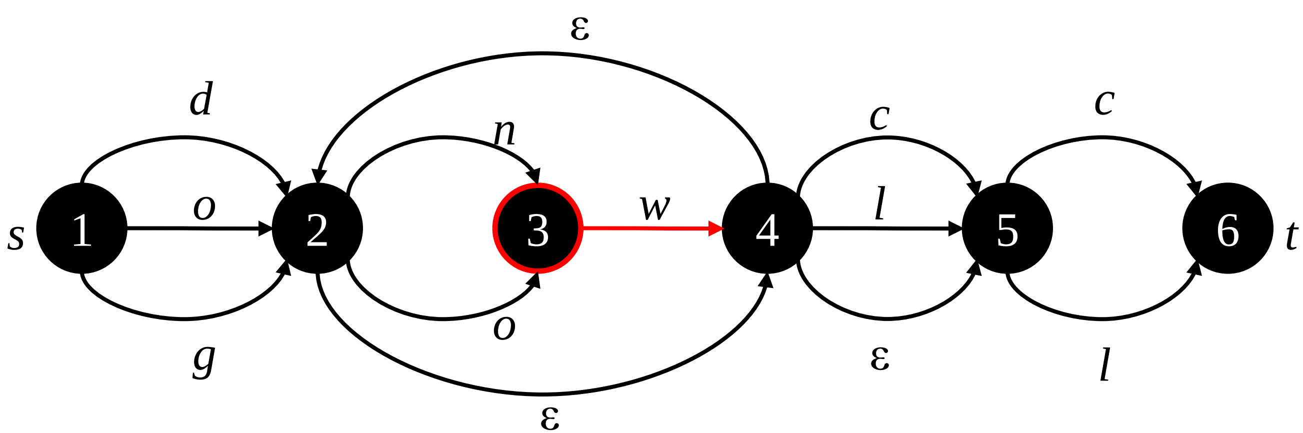
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3			

Failure functions

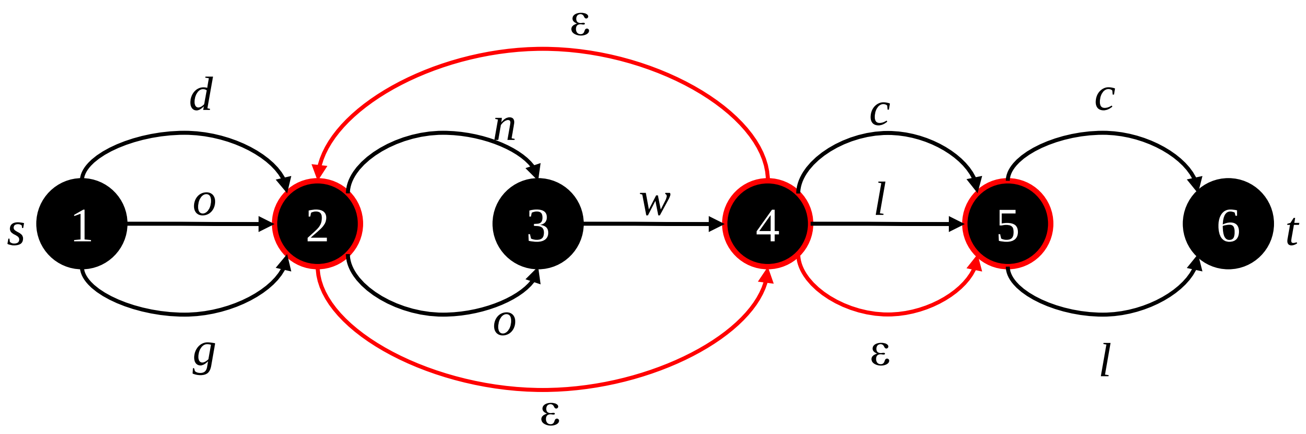
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3			

Failure functions

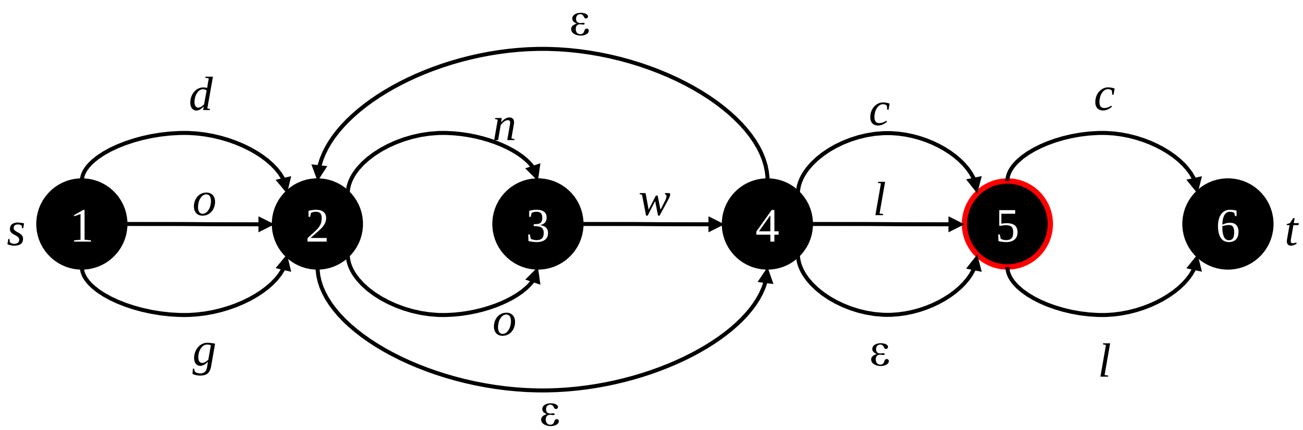
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5		

Failure functions

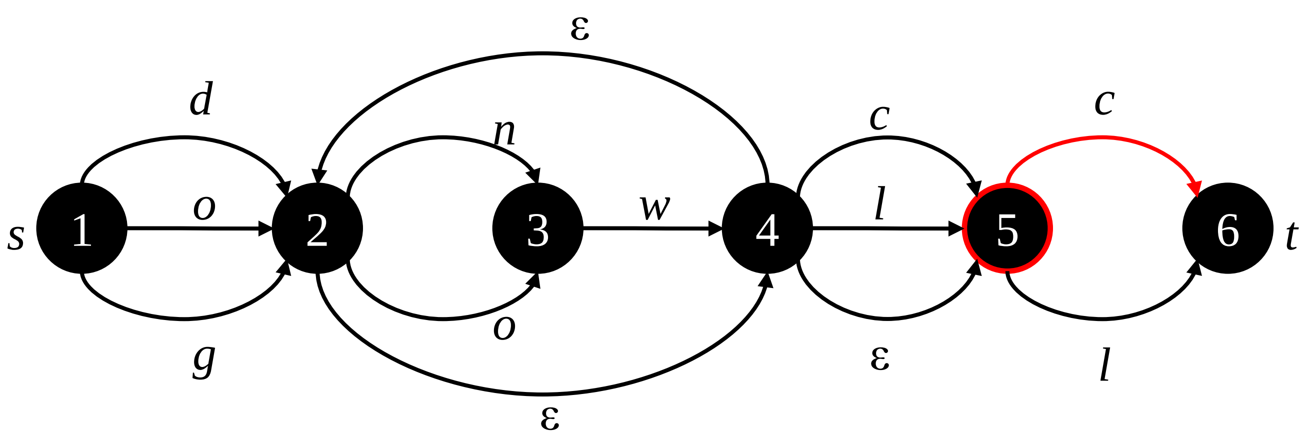
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5	5, 6	

Failure functions

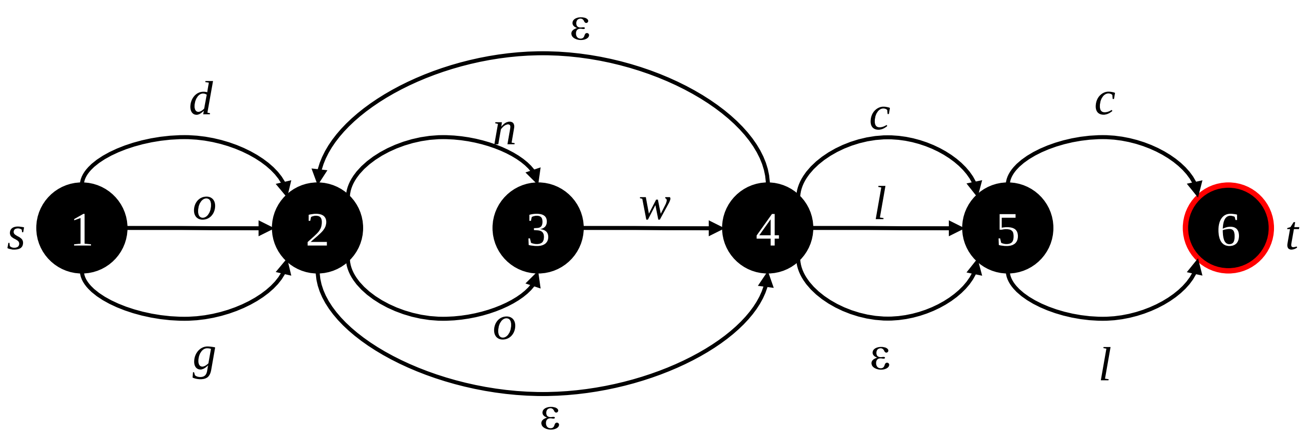
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5	5, 6	

Failure functions

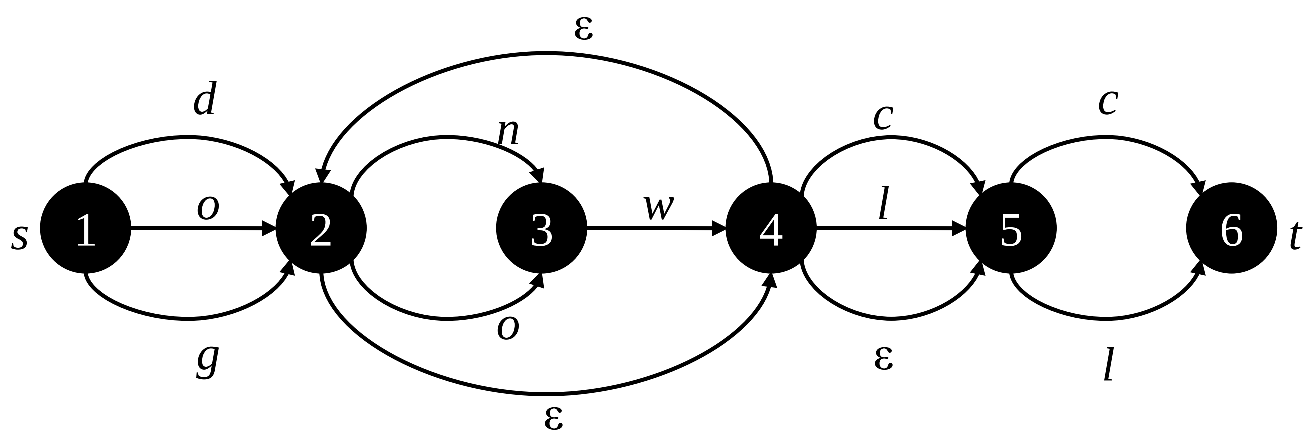
- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5	5, 6	6

Failure functions

- **Directed graph**

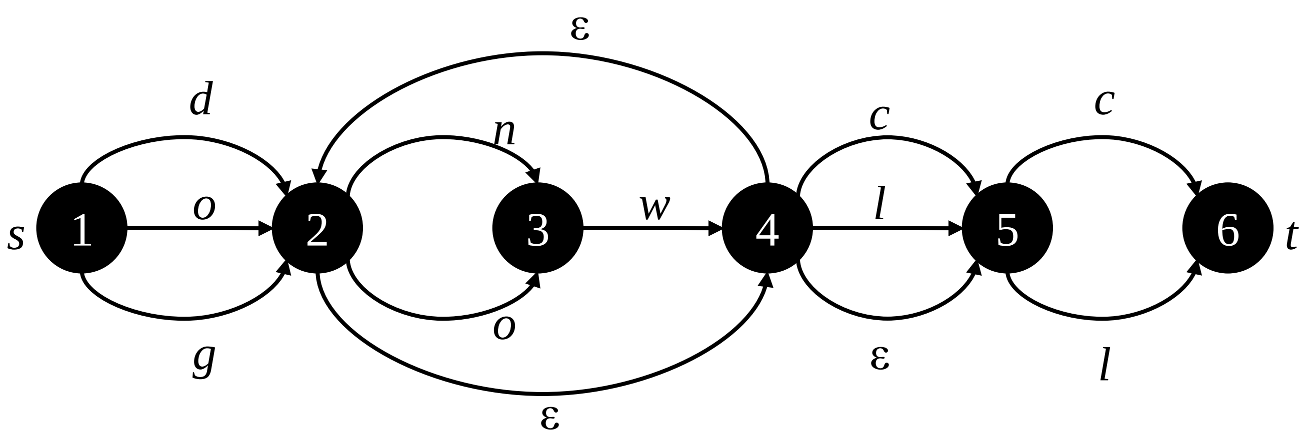


i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5	5, 6	6

- To find all **prefixes of T that match** *R*, compute the sets *N(i)* for *i* from 0 to n, the length of T

Failure functions

- **Directed graph**

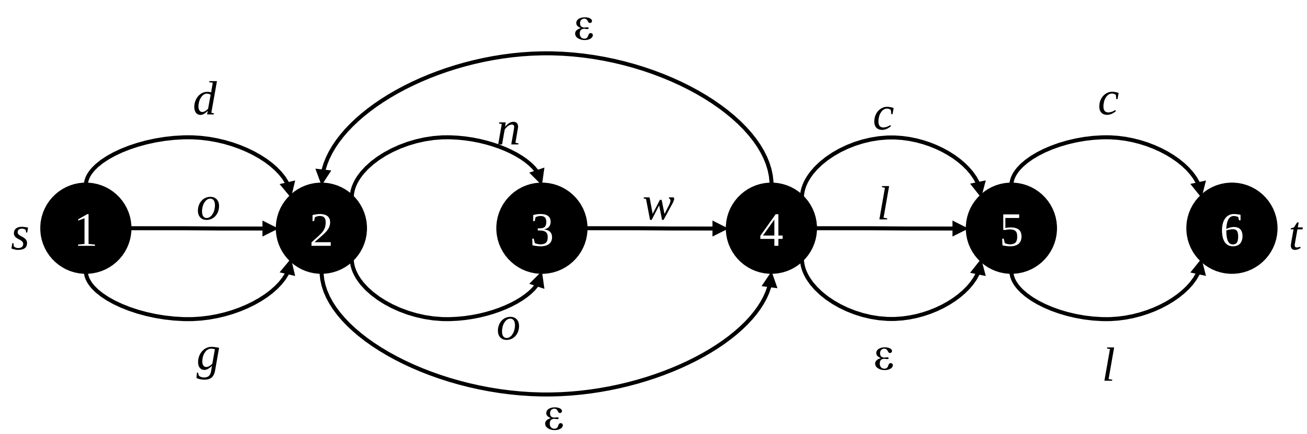


i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5	5, 6	6

- If *G(R)* contains *e* edges, then the time for this algorithm is ***O(ne)***

Failure functions

- **Directed graph**



i	0	1	2	3	4	5	6	7
$T(i)$		d	n	w	o	w	c	l
$N(i)$	1	2, 4, 5	3	2, 4, 5	3	2, 4, 5	5, 6	6

- If a regular expression R has m symbols, then $G(R)$ can be constructed using at most $2m$

Formal definitions

- **Theorem 3.6.1**
  - If T is of length n, and the regular expression R contains m symbols, then it is possible to determine whether T contains a substring match R in O(nm) time.