Some tips for preparation

- Solve as many problems from previous exams as you have time for. Even if you feel you already know how to solve a problem, practice helps you improve your speed, which matters on the exam.
- Understand the inner workings and invariants of algorithms (e.g. what various algorithms look like if stopped while executing).
- It is difficult to remember the running times of all the algorithms. Use your sheet of notes.
- Feel free to use Piazza to share suggestions on what to put on the notes page.

Some tips for solving problems

- Key technique: figuring out how best to represent the data. The right data structure may not be obvious.
- In particular, graphs are a powerful data type and may be applicable in design problems even if the wording doesn't immediately suggest it.
- Max-flow is a powerful technique for all kinds of optimization problems (e.g. optimal matching of students to companies).
- Key technique: If a problem is a slight variant of a known problem, try to transform it into an instance of the known problem.

How do you feel about the final?

A. 😃

B. 😐

C. 🙁

D. 😬

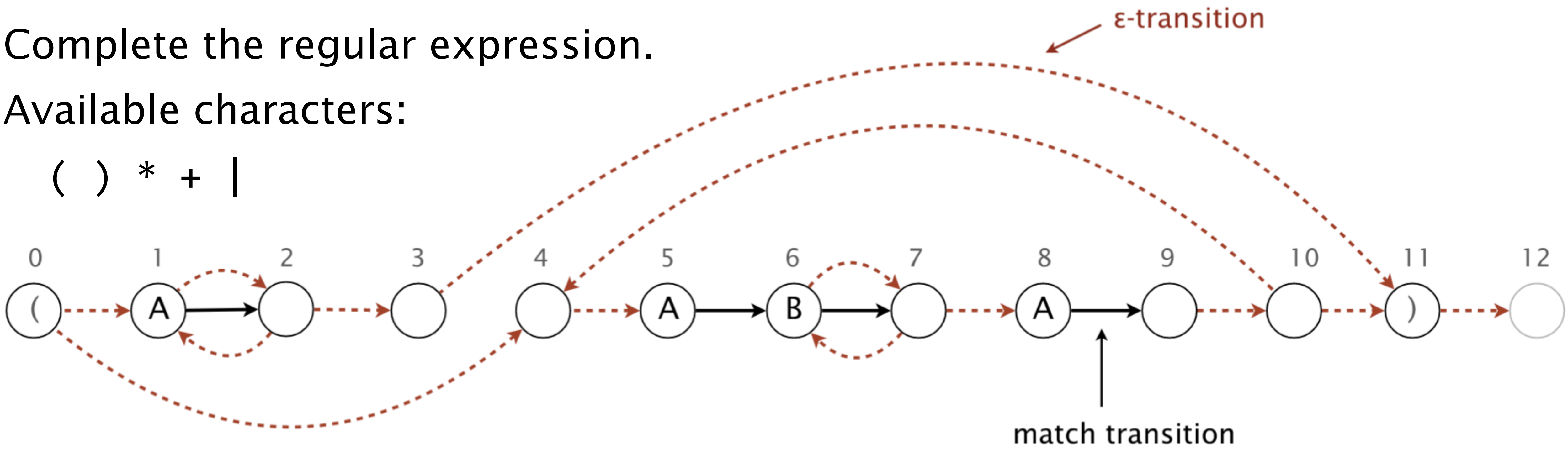
E. 🤷‍♀️

NFA [Fall 2017; easy]

Complete the regular expression.

Available characters:

( ) * + |

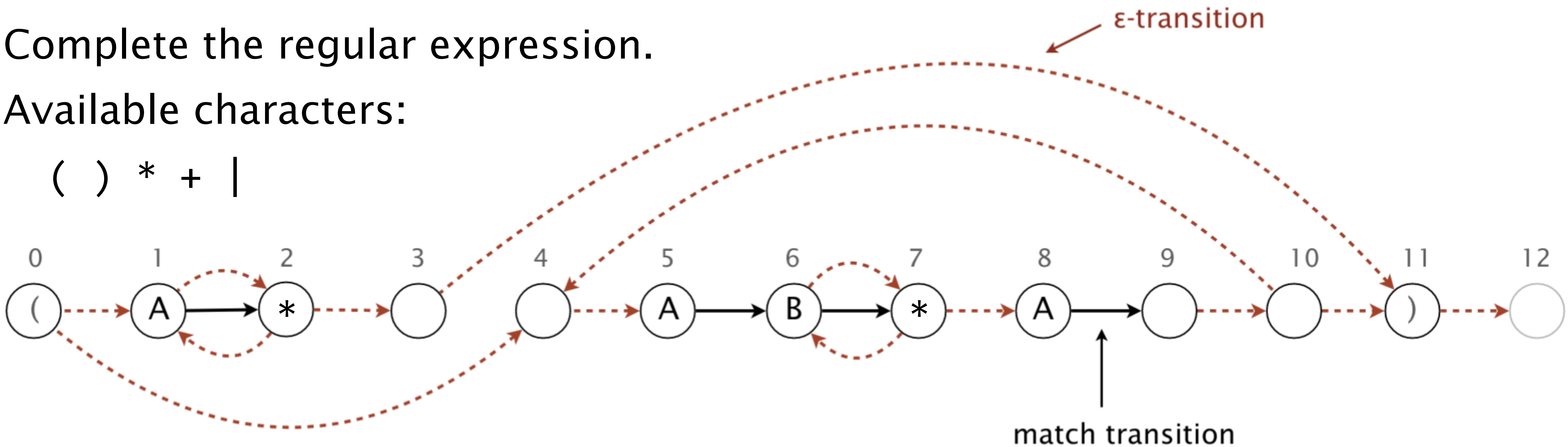


NFA [Fall 2017; easy]

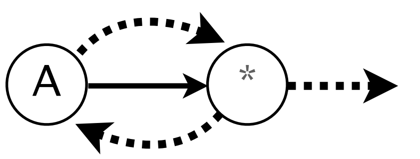
Complete the regular expression.

Available characters:

( ) * + |



- Familiar pattern 1:

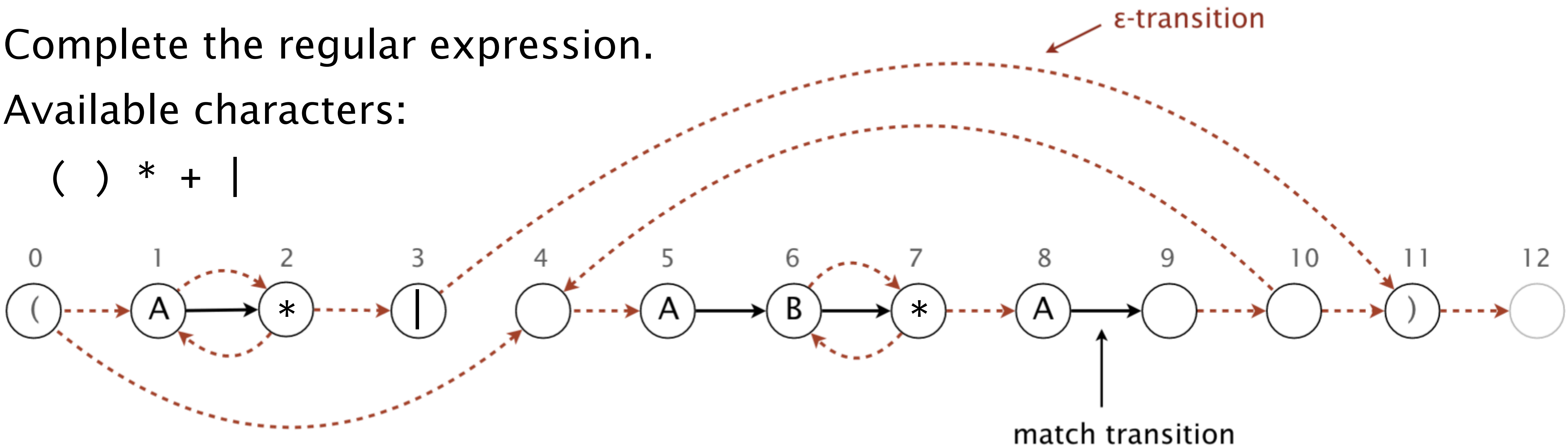


NFA [Fall 2017; easy]

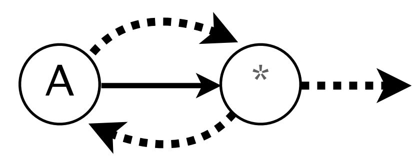
Complete the regular expression.

Available characters:

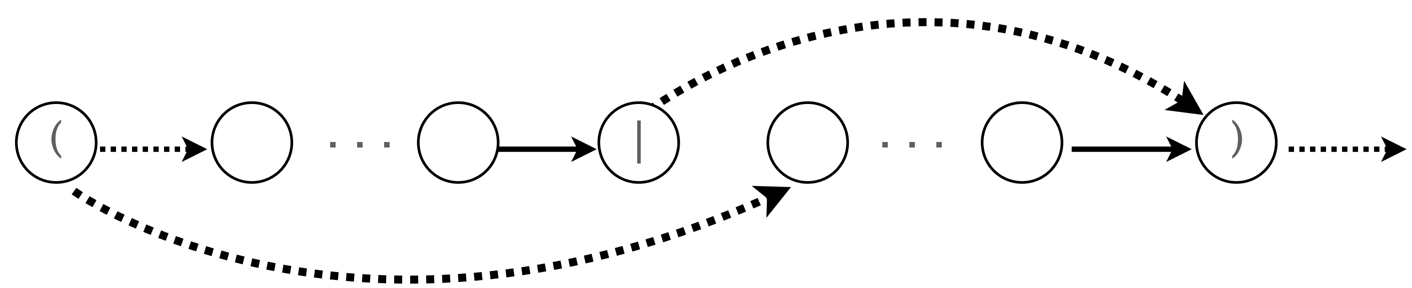
( ) * + |



- Familiar pattern 1:



- Familiar pattern 2:

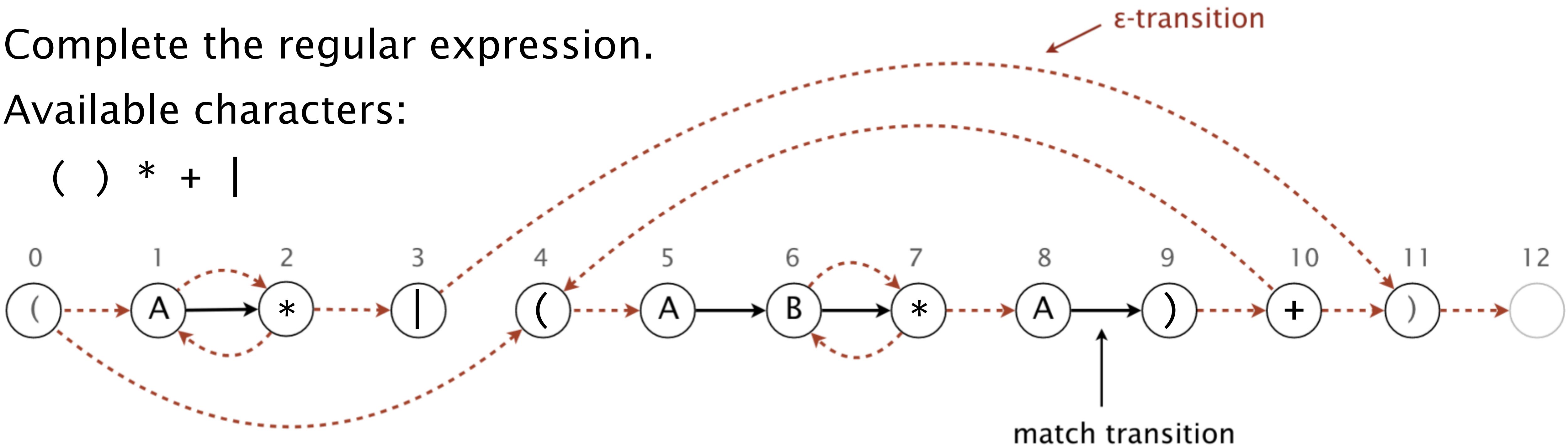


NFA [Fall 2017; easy]

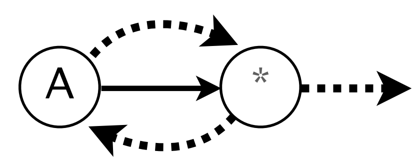
Complete the regular expression.

Available characters:

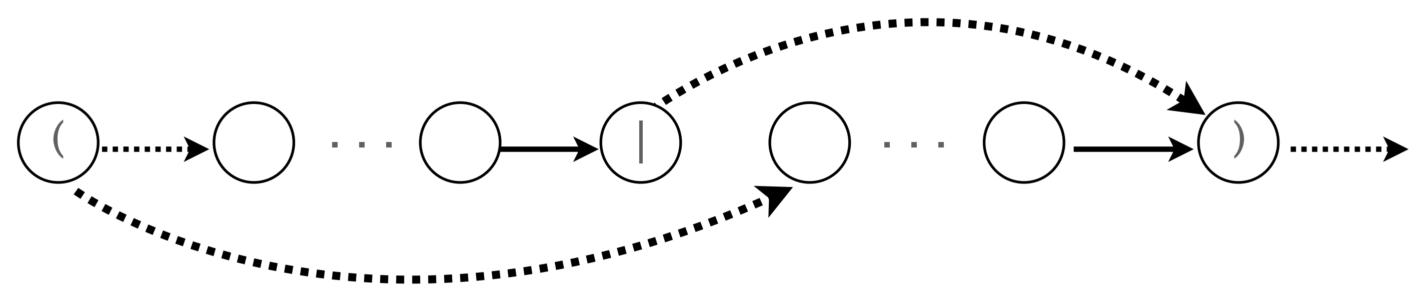
( ) * + |



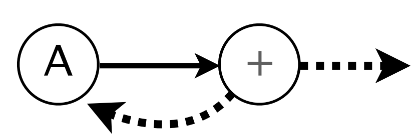
- Familiar pattern 1:



- Familiar pattern 2:



- Slightly less familiar pattern 3:



DFA [Spring 2016; crazy]

Here is a partially completed KMP DFA over the alphabet `{A, B, C}`. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			
C		0				3

List the string that the DFA searches for: _ _ _ _ _ _

DFA [Spring 2016; crazy]

Here is a partially completed KMP DFA over the alphabet `{A, B, C}`. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			
C		0				3

List the string that the DFA searches for: _ _ _ _ _ _

Hints:

- Fill in the table and the string in parallel.
- Look at the last column. How to go from state 5 to 6? So last char is…?
- Similarly, how to go from state 2 to 3?
- `dfa[2]['B'] = 1.` So the first character of the string is… ?
- `dfa[5]['C'] = 3.` So how are the first & second halves of the string related?
- Almost done! Figure it out by elimination.

DFA [Spring 2016]

Here is a partially completed KMP DFA over the alphabet `{A, B, C}`. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			**6**
C		0				3

List the string that the DFA searches for: _ _ _ _ _ B

DFA [Spring 2016]

Here is a partially completed KMP DFA over the alphabet {A, B, C}. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			**6**
C		0	**3**			3

List the string that the DFA searches for: _ _ C _ _ B

DFA [Spring 2016]

Here is a partially completed KMP DFA over the alphabet `{A, B, C}`. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			**6**
C		0	**3**			3

List the string that the DFA searches for: B _ C _ _ B

'1' can appear only in the row corresponding to the first character of pattern.

DFA [Spring 2016]

Here is a partially completed KMP DFA over the alphabet `{A, B, C}`. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			**6**
C		0	**3**			3

List the string that the DFA searches for: B _ C _ _ B

`dfa[5]['C'] = 3`. So the string is BxCBxB.

x can't be `C` — if it were, `dfa[1]['C']` would be 2.

x can't be `B` — if it were, `dfa[2]['B']` would be 2.

DFA [Spring 2016]

Here is a partially completed KMP DFA over the alphabet `{A, B, C}`. State 6 is the accept state. Fill in the missing cells.

j	0	1	2	3	4	5
A			0			0
B			1			**6**
C		0	**3**			3

List the string that the DFA searches for: B A C B A B

Now complete the table using the familiar DFA construction algorithm.

Key technique: finding the right data type / data structure.

Given a list of valid words, preprocess the list so that you can use it to solve word ladder problems efficiently (e.g. transform FOOL to SAGE by changing one character at a time).

Note: if this were an exam problem the wording of the question would be much more precise.

First guess: trie? Think again!

FOOL
POOL
POLL
POLE
PALE
SALE
SAGE

Key technique: finding the right data type / data structure.

Given a list of valid words, preprocess the list so that you can use it to solve word ladder problems efficiently (e.g. transform FOOL to SAGE by changing one character at a time).

Note: if this were an exam problem the wording of the question would be much more precise.

First guess: trie? Think again!

From FOOL we can go to FOOT, FOAL, ... words that are "close". "Adjacent", in a sense. Oh, so a graph of words!

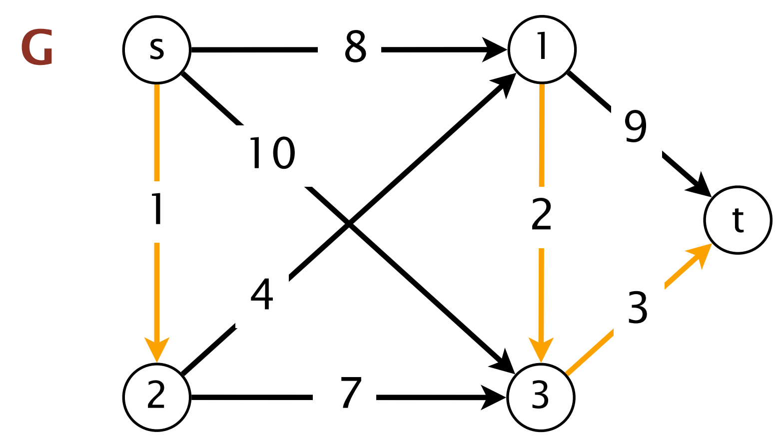
FOOL
POOL
POLL
POLE
PALE
SALE
SAGE

To find a word ladder, run BFS from the source word.

Shortest path with orange and black edges

Goal. Given a digraph, where each edge has a positive weight and is orange or black, find shortest path from s to t that uses at most k orange edges.

Key technique. If a problem is a slight variant of a known problem, try to transform it into an instance of the known problem.



k = 0: s→1→t (17)

k = 1: s→3→t (13)

k = 2: s→2→3→t (11)

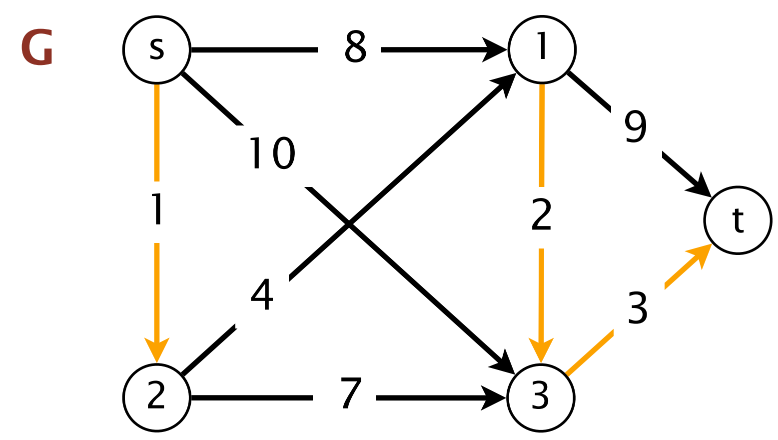
k = 3: s→2→1→3→t (10)

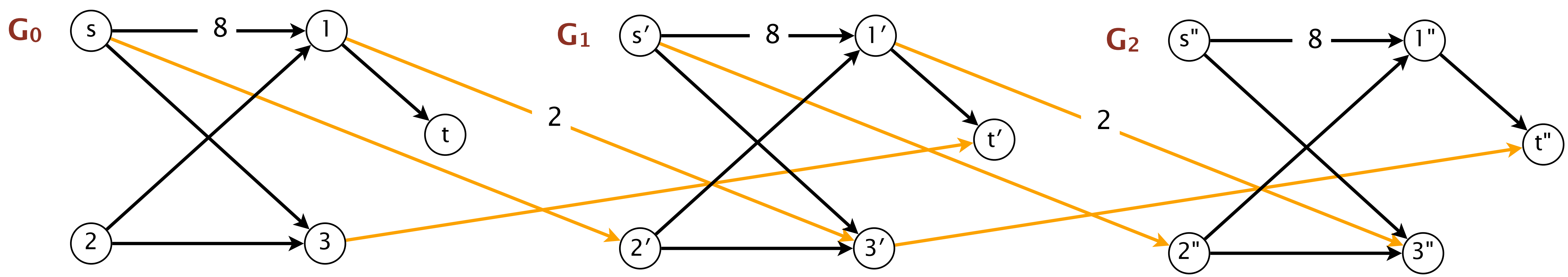
Shortest path with orange and black edges

Goal. Given a digraph, where each edge has a positive weight and is orange or black, find shortest path from s to t that uses at most k orange edges.

Solution. Create $k+1$ copies of the digraph $G_0, G_1, \ldots, G_k$. For each edge $v \rightarrow w$

- Black: add edge from vertex v in graph G_i to vertex w in G_i.
- Orange: add edge from vertex v in graph G_i to vertex w in G_{i+1}.



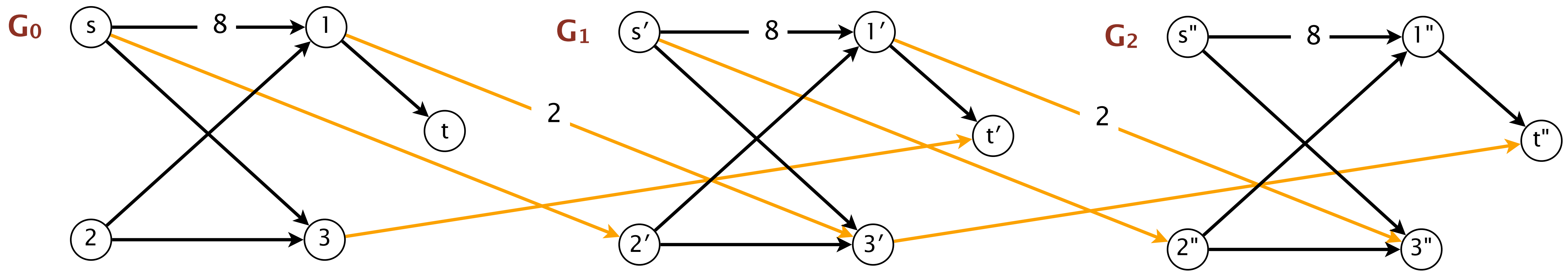


Shortest path with orange and black edges

Goal. Given a digraph, where each edge has a positive weight and is orange or black, find shortest path from s to t that uses at most k orange edges.

Solution. Create $k+1$ copies of the digraph $G_0, G_1, \ldots, G_k$. For each edge $v \rightarrow w$

- Black: add edge from vertex v in graph G_i to vertex w in G_i.
- Orange: add edge from vertex v in graph G_i to vertex w in G_{i+1}.
- Find shortest path from s to every copy of t (and choose best).



Slight tweaks to standard algorithmic problems [Fall 2014]

Given an edge-weighted digraph in which all edge weights are either 1 or 2 and two vertices s and t, find a shortest path from s to t in time proportional to $E + V$.

Given an edge-weighted DAG with positive edge weights and two vertices s and t, find a *path* from s to t that *maximizes the product* of the weights of the edges participating in the path in time proportional to $E + V$.

Given an array of N strings over the DNA alphabet $\{A, C, T, G\}$, determine whether all N strings are distinct in time linear in the number of characters in the input.

Given an array a of N 64-bit integers, determine whether there are two indices i and j such that $a_i + a_j = 0$ in time proportional to N.

Given an array of N integers between 0 and $R^2 - 1$, *stably sort* them in time proportional to $N + R$.