Language & Technology

Lecture 6: More on Spell Checking

Thomas Graf

Stony Brook University
lin425@thomasgraf.net

What we Know About Spell Checking So Far

- dictionary as a prefix tree
- spell checking = lookup in prefix tree
  - word found $\Rightarrow$ spelled correctly
  - word not found $\Rightarrow$ spelled incorrectly
- But this simple model is **not good enough**.

Open Issues

- Not all misspellings are easy to detect.
- How do we determine the correct spelling of a mistyped word?
- What if a correctly spelled word is not in the dictionary?
  specialized terminology, proper names, neologisms, slang, loan words, ...

What we Know About Spell Checking So Far

- dictionary as a prefix tree
- spell checking = lookup in prefix tree
  - word found ⇒ spelled correctly
  - word not found ⇒ spelled incorrectly
- But this simple model is **not good enough**.

Open Issues

- Not all misspellings are easy to detect.
- How do we determine the correct spelling of a mistyped word?
- What if a correctly spelled word is not in the dictionary?
  specialized terminology, proper names, neologisms, slang, loan words, ...

Assessing the Problem

- Never type a single line of code before you understand the problem!
- Think about the **parameters of the problem**.
- Solving the wrong problem is pointless.

Parameters of the Spell Checking Problem

- How is the spell checker to be used?
  automatic/auto correction VS interactive/suggestions to user
- What types of misspellings are there?
- Is it feasible to detect all of them?
- Once we know what the tool should handle,
  what is the simplest solution?

Assessing the Problem

- Never type a single line of code before you understand the problem!
- Think about the **parameters of the problem**.
- Solving the wrong problem is pointless.

Parameters of the Spell Checking Problem

- How is the spell checker to be used?
  automatic/auto correction VS interactive/suggestions to user
- What types of misspellings are there?
- Is it feasible to detect all of them?
- Once we know what the tool should handle,
  what is the simplest solution?

A Typology of Spelling Mistakes

- **Cause**
  accidental typo $\Leftrightarrow$ spelling confusion/unawareness of correct spelling
- **Number**
  single-error $\Leftrightarrow$ multi-error
- **Error Type**

split	illicit space
	quin tuplets, the set up, atoll way
run-on	missing space
	nightvision, boothbabe, atoll way
non-word	typed word does not exist
	warte or tawer for water
real-word	misspelling yields existing word(s) of English
	car toon, it's VS its, their VS there, book for brook, atoll way

A Difficulty Hierarchy of Tasks

- Both typos and spelling confusion can be very hard.
  - *labelled* for *labeled*: easy
  - *awter* for *water*: easy
  - *nitch* for *niche*: tricky
  - *awre* for *water*: tricky
- Single-error is easier than multi-error.
- Non-word errors are easier than real-word errors.
- Difficulty of real-word errors scales with complexity of context:
  - local syntactic configuration
  - non-local syntactic configuration
  - word meaning
  - discourse/cross-sentence
  - world knowledge

Examples of Increasing Context Complexity

- **Local Syntactic Configuration**
  Their are some biscuits on the counter.
- **Non-Local Syntactic Configuration**
  The man sitting at the bar seem to be enjoying the atmosphere.
- **Word Meaning**
  We still have to pay off the mortgage on our mouse.
- **Discourse**
  It's like that time they canceled Futurama. I was so bad.
- **World Knowledge**
  This course is taught at Stony Book University.

How Much Can We Handle Efficiently?

- Remember what we've said many times before:
  meaning is hard, world knowledge nigh impossible.
- Even non-local syntactic configurations are difficult to deal with.
- So we consider only models that handle at most
  local syntactic configurations.

n-Grams Handle Local Context

- *n*-gram models can easily detect local real-word errors.

Example

- No English sentence ever contains the bigram *their are*.
- Consequently, every instance of *their are* must be a misspelling.

Problems:

- Still problem with correct words that are not listed in dictionary.
- For automatic spell checker, what about cases like *the man are*?
  change *man* to *men* VS change *are* to *is*
- For interactive spell checker, how do we find all relevant suggestions?

n-Grams Handle Local Context

- *n*-gram models can easily detect local real-word errors.

Example

- No English sentence ever contains the bigram *their are*.
- Consequently, every instance of *their are* must be a misspelling.

Problems:

- Still problem with correct words that are not listed in dictionary.
- For automatic spell checker, what about cases like *the man are*?
  change *man* to *men* VS change *are* to *is*
- For interactive spell checker, how do we find all relevant suggestions?

Unlisted Words are Inevitable

- It is impossible to have every word of English listed in the dictionary.
- It is also **undesirable**:
  - No speaker knows or uses all words of English.
  - Suppose the user knows only 10% of the words in our dictionary. Then there is a good chance that a non-word error by the user is a real-word error for our model.
  - **Bottom line**
    The bigger the dictionary, the harder it is to detect misspellings.

Example

- Computer scientists use the special term *memoize*.
- But for most people *memoize* is a misspelling of *memorize*.
- If the dictionary contains *memoize*, the model performs **worse** for the majority of users.

Unlisted Words are Inevitable

- It is impossible to have every word of English listed in the dictionary.
- It is also **undesirable**:
  - No speaker knows or uses all words of English.
  - Suppose the user knows only 10% of the words in our dictionary. Then there is a good chance that a non-word error by the user is a real-word error for our model.
  - **Bottom line**
    The bigger the dictionary, the harder it is to detect misspellings.

Example

- Computer scientists use the special term *memoize*.
- But for most people *memoize* is a misspelling of *memorize*.
- If the dictionary contains *memoize*, the model performs **worse** for the majority of users.

A Cool Idea That is Hard to Realize

- One conceivable solution is to **stratify** the dictionary into
  - a base vocabulary used by all English speakers, and
  - optional extensions for specific genres, styles, etc.
- Extensions could be loaded if the text so far fits certain criteria.
  high number of field-specific terms, loan words, etc.
- To the best of my knowledge, nobody has ever tried anything like this.
- The payoff probably isn't worth the effort.

Another Solution for Unlisted Words

- Humans can easily distinguish possible words of English from impossible ones.

Example

possible	**impossible**
blick	bnick
wrexel	rwexel
lakoo	ooakl
orcalate	orclte

- Only certain sequences of characters can occur in English words.
- You guessed it: **n-grams again!**

Character *n*-Grams for Non-Word Detection

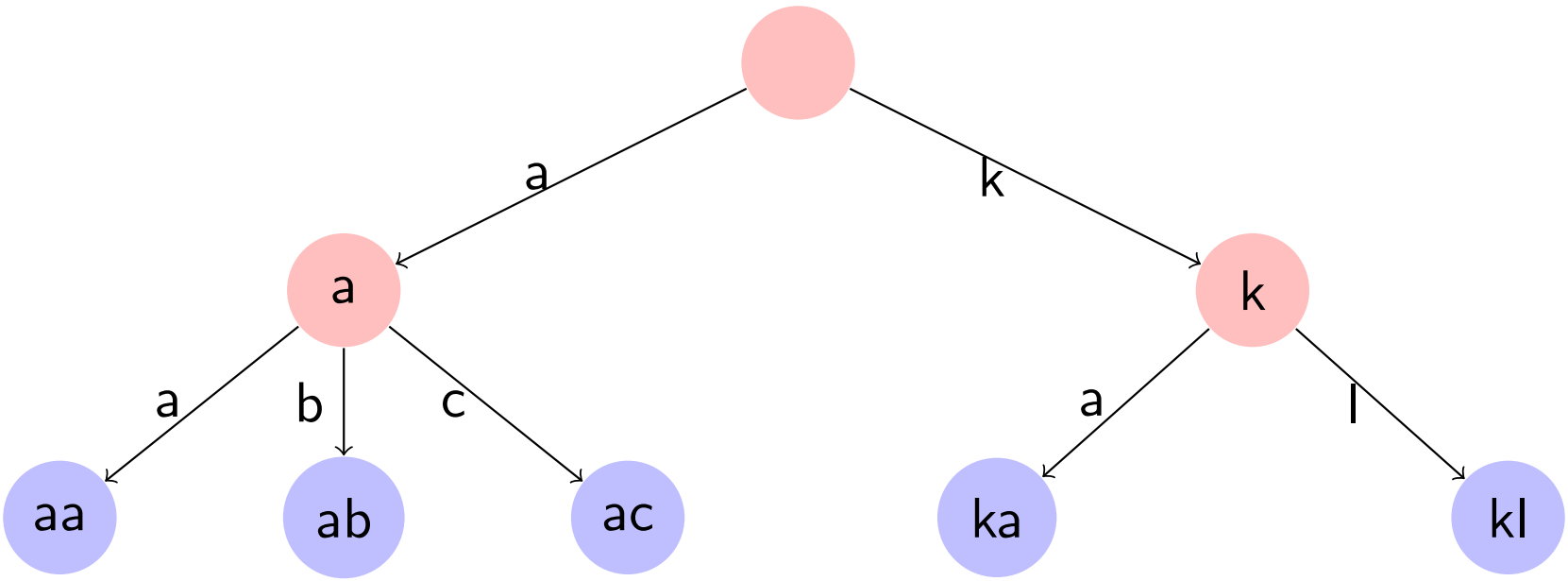
Non-Word Detection Algorithm

- Compile list of character bigrams that occur in words in dictionary.
- If a word
  1. is not in the dictionary, and
  2. contains an illicit character bigram

  it is a non-word.

Excursus: Storing Character Bigram Lists

- The list of licit character bigrams could be stored as a prefix tree.
- But prefix trees are a little odd in this case:
  - only two levels deep
  - no use for good/bad distinction (i.e. red and blue)



Excursus: Storing Character Bigram Lists [cont.]

- Simpler solution: **matrix/array**
- Rows and columns labeled by characters.
- If xy is a licit character bigram, enter 1 in the cell in row x and column y.

	···	k	l	m	···
⋮					
k		1	1	0	
l		1	1	1	
m		0	0	1	
⋮					

Relevant examples: akkadian, backlog, walk, wall, balm, hammock

If the matrix has only a few 1-cells, you're better off using a simple list.

Which Impossible Words are Detected?

- We have seen several times by now that bigrams are insufficient in certain applications.
- Instead, trigrams or maybe even 4-grams or 5-grams may be necessary.
- Depending on how much accuracy we need, this is also the case here.

Example

Impossible Word	**Character Bigrams**	**Illicit Bigrams**
bnick	bn, ni, ic, ck	bn
rwexel	rw, we, ex, xe, el	**none!**
akklaim	ak, kk, kl, la, ai, im	**none!**

Improving the *n*-Gram Model

1. **Size**
   - trigrams much better than bigrams
   - *Example*
     kk and *kl* occur in a few English words, but not *kkl*
2. **End Markers**
   - beginning and end of English words are special
   - *Examples*
     nk is very common, but impossible at beginning of word
     cl is very common, but impossible at end of word
   - special character $ for word edges
     $nk impossible, *nk* and *nk$* allowed
3. **Probabilities**
   - determine frequency of character *n*-grams
   - treat non-word probability as product of character *n*-grams
   - everything below a certain threshold is a non-word

What Kind of Spell Checker is This Good For?

Finding the Correct Word

- We now have a way of dealing with unknown words.
- **But:** still need mechanism for spelling suggestions.
- **Three Common Approaches**
  - rule-based
  - similarity key
  - minimum edit distance

Option 1: Ruled-Based Approach

- custom rules that provide candidate lists for specific misspellings

Example

- teh $\Rightarrow$ the
- refering $\Rightarrow$ referring
- fyre $\Rightarrow$ fire, fry

- can be collected by hand or automatically

Advantages	**Disadvantages**
conceptually simple	labor intense
	inflexible

Option 1: Ruled-Based Approach

- custom rules that provide candidate lists for specific misspellings

Example

- teh ⇒ the
- refering ⇒ referring
- fyre ⇒ fire, fry

- can be collected by hand or automatically

Advantages	**Disadvantages**
conceptually simple	labor intense
	inflexible

Option 2: Similarity Keys

- classify strings of characters according to similarity
- words in same similarity class are offered as suggestions

Example: US Census Soundex Algorithm

designed for indexing names by sound

1. Retain first letter.
2. Drop all (other) occurrences of *a*, *e*, *i*, *o*, *u*, *y*, *h*, *w*.
3. Replace consonsants by digits:
   - *b*, *f*, *p*, *v* $\Rightarrow$ 1
   - *c*, *g*, *j*, *k*, *s*, *x*, *z* $\Rightarrow$ 2
   - *d*, *t* $\Rightarrow$ 3
   - *l* $\Rightarrow$ 4
   - *m*, *n* $\Rightarrow$ 5
   - *r* $\Rightarrow$ 6
4. Remove consecutive copies of same digit.
5. Shorten/lengthen to 4 characters.

bearded	$\xrightarrow{1+2}$
brdd	$\xrightarrow{3}$
b633	$\xrightarrow{4}$
b63	$\xrightarrow{5}$
b630	$\xleftarrow{5}$
b63	$\xleftarrow{4}$
b663	$\xleftarrow{3}$
brrd	$\xleftarrow{1+2}$
borrowed	

(Dis)Advantages of Similarity Keys

Advantages

- easy to implement
- easy to compute

Disadvantages

- similarity key must be carefully designed for application
  Soundex is not a good choice for spell checking
- may need distinct key for distinct languages

Option 3: Minimum Edit Distance

How many steps does it take to transform one word into another?

Levenshtein Distance

The Levenshtein Distance between x and y is n if the shortest sequence of

- single character deletions, and/or
- single character insertions, and/or
- single character substitutions

that turns x into y takes n steps.

Example: Transforming *meat* into *bats*

String	Operation
meat	
beat	substitute *b* for *m*
bat	delete *e*
bats	insert *s*

Computing Levenshtein Distances

- The Levensthein distance is determined by the **shortest sequence** of operations.
- How do we know that there isn't a shorter solution?

Naive Solution

- Try all possible 1-step sequences.
- If desired word among outputs, Levenshtein distance is 1.
- Otherwise, try all 2-step sequences.
- If desired word among outputs, Levenshtein distance is 2.
- Otherwise, . . .

Evalutating the Naive Solution

- The naive solution is guaranteed to terminate (= it won't run forever).
- **Reason:** The Levenshtein distance between two words is at most the length of the longer one.
- **But:** The **combinatorial explosion** is enormous.

Example

4 character word, 26 letter alphabet

Steps	**Possible Operations**	**Computed Strings**
1	4 del, 5×26 ins, 4×26 sub	238
2	10 del, 15×26 ins, 10×26 sub	660

Improving Efficiency

- **Problem 1:** "generate and test" is too undirected
- **Problem 2:** n-step calculation repeats computations from $(n-1)$-step calculation

Dynamic Programming

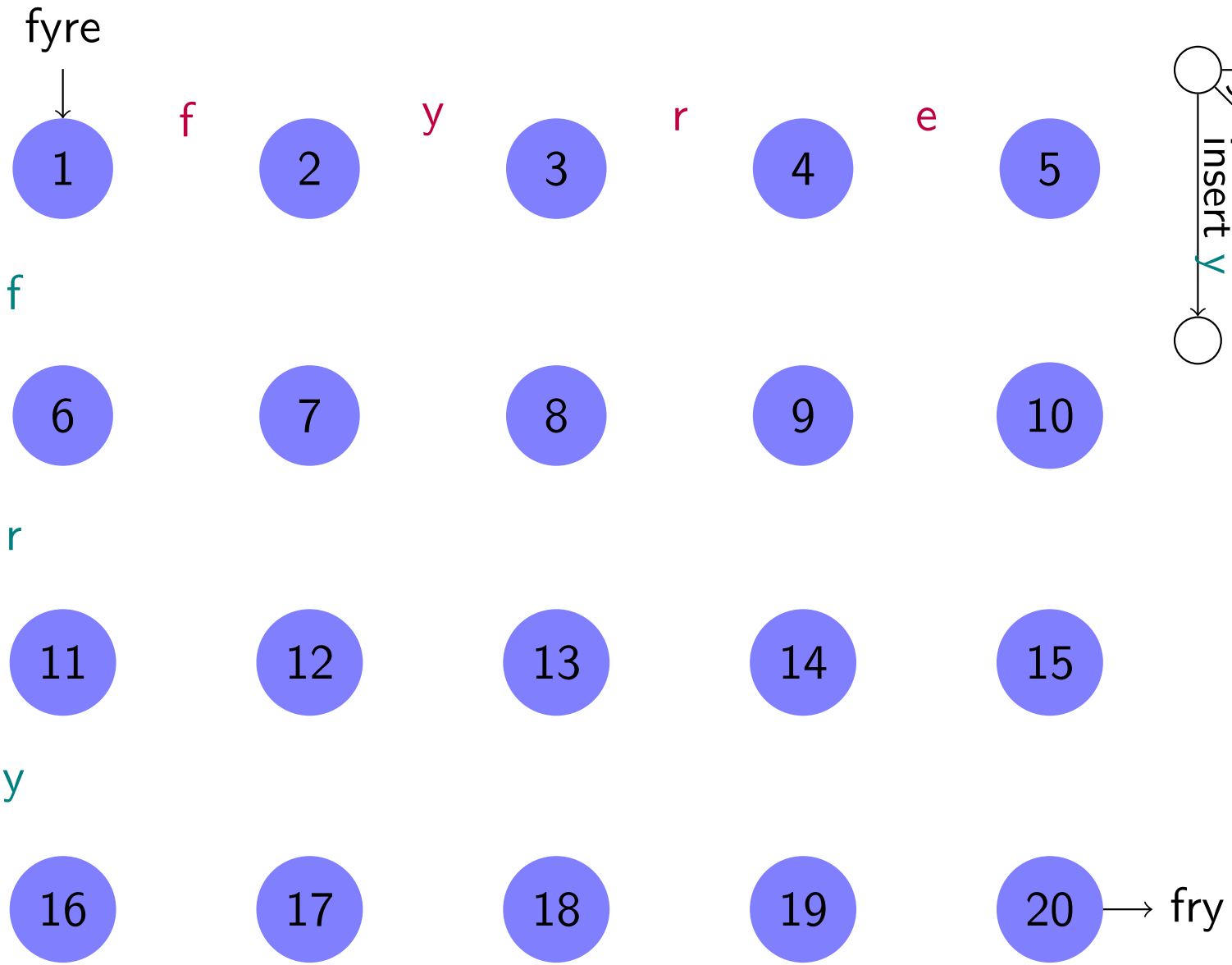
1. Decompose big problems into small problems.
2. Solve the small problems and **save the solution**.
3. Look up stored solutions rather than recomputing them.

Intuition: Write down intermediate results, just like humans do.

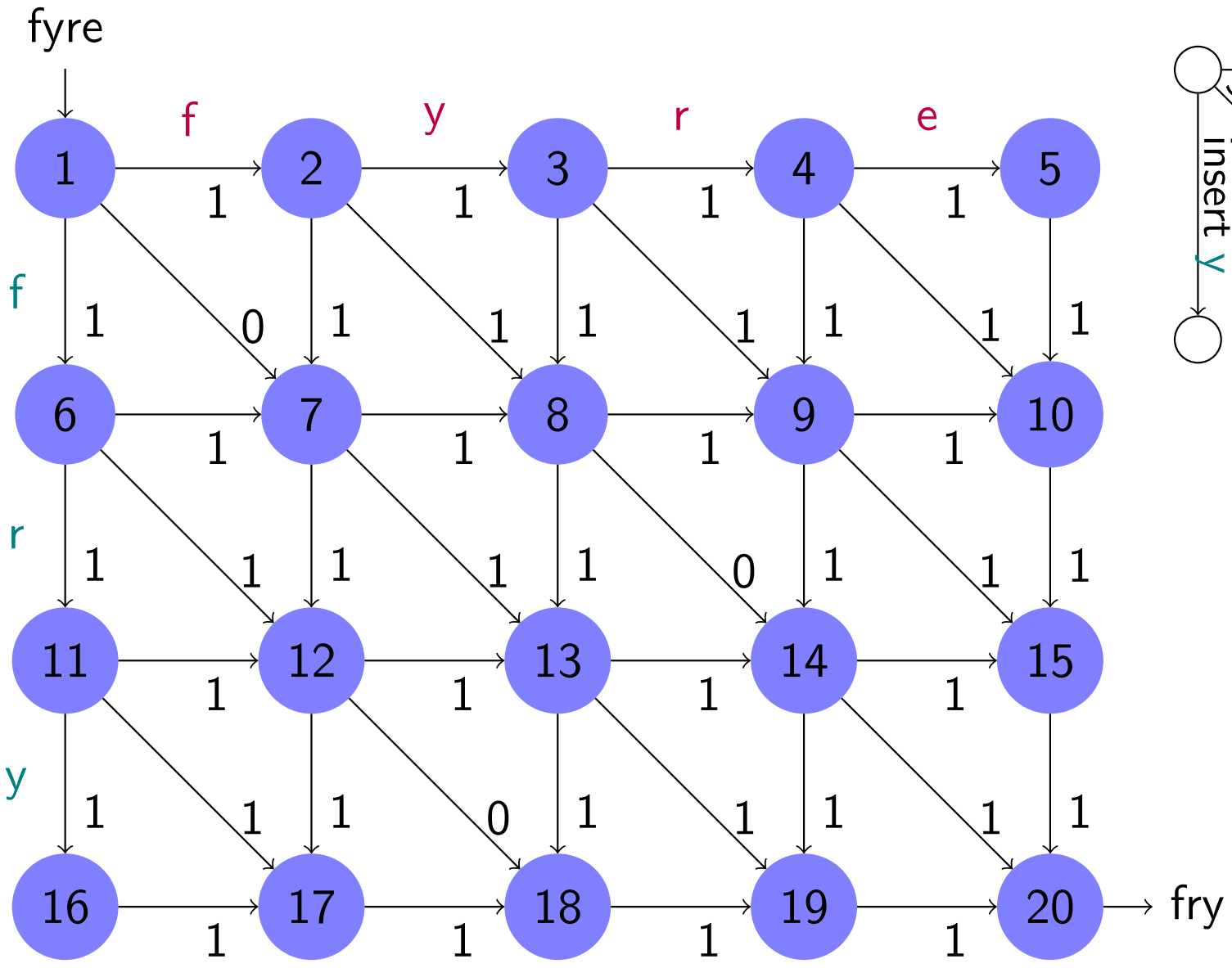
Dynamic Programming for Levenshtein Distance

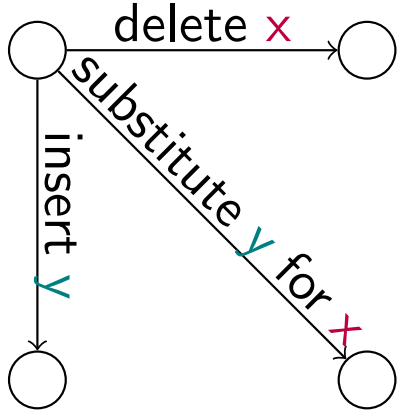
- Draw a graph that represents the possible edit sequences from x to y.
- We want the least costly path through that graph.
- Dynamic programming solution:
  1. For each node, what is the least costly path to it from adjacent nodes?
  2. Throw away all other paths.
  3. Go backwards from target to source to find correct path.

Example of Dynamic Programming

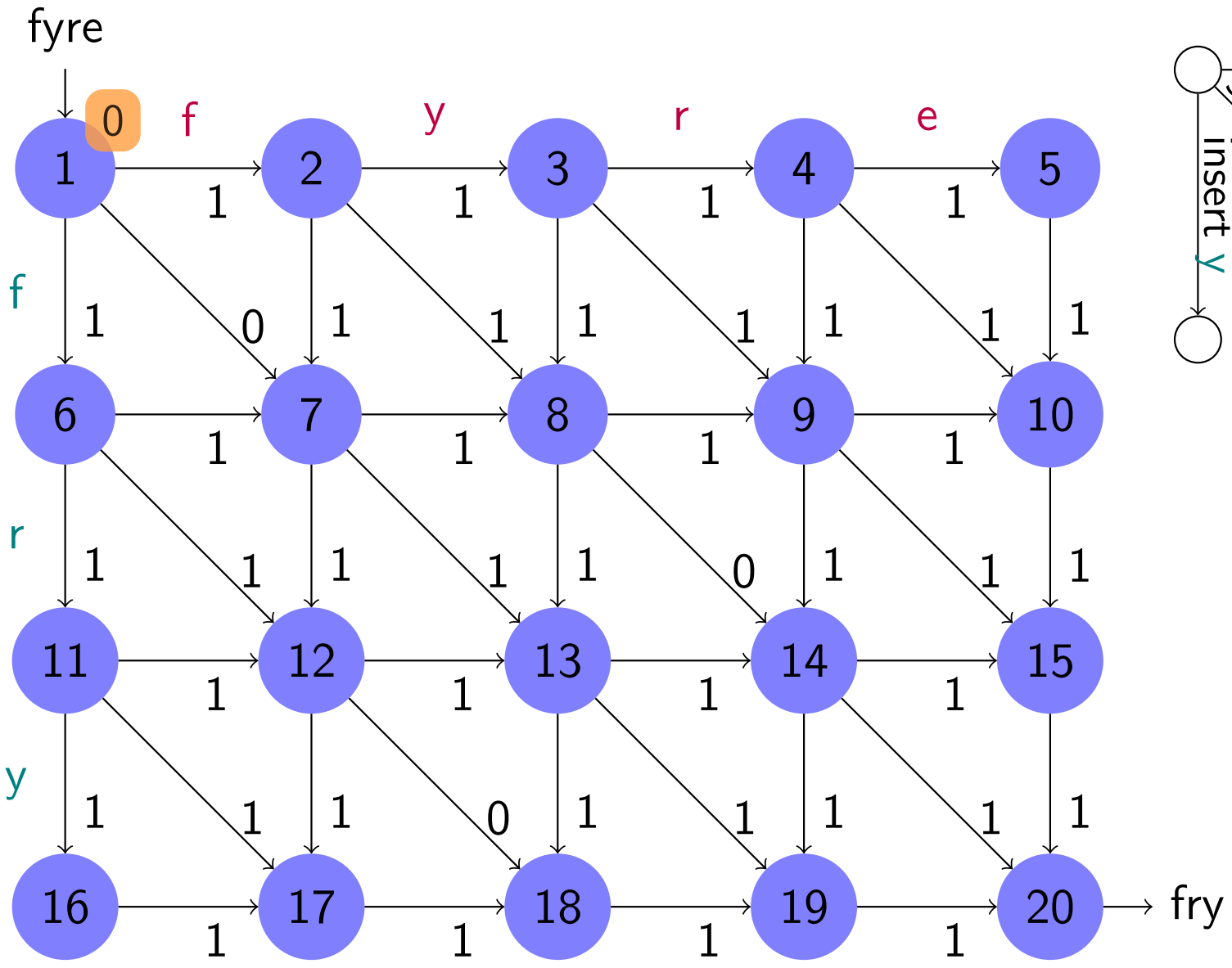


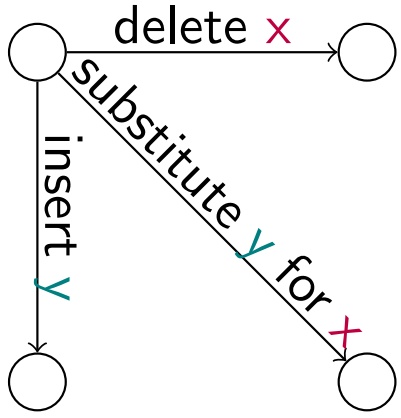
Example of Dynamic Programming



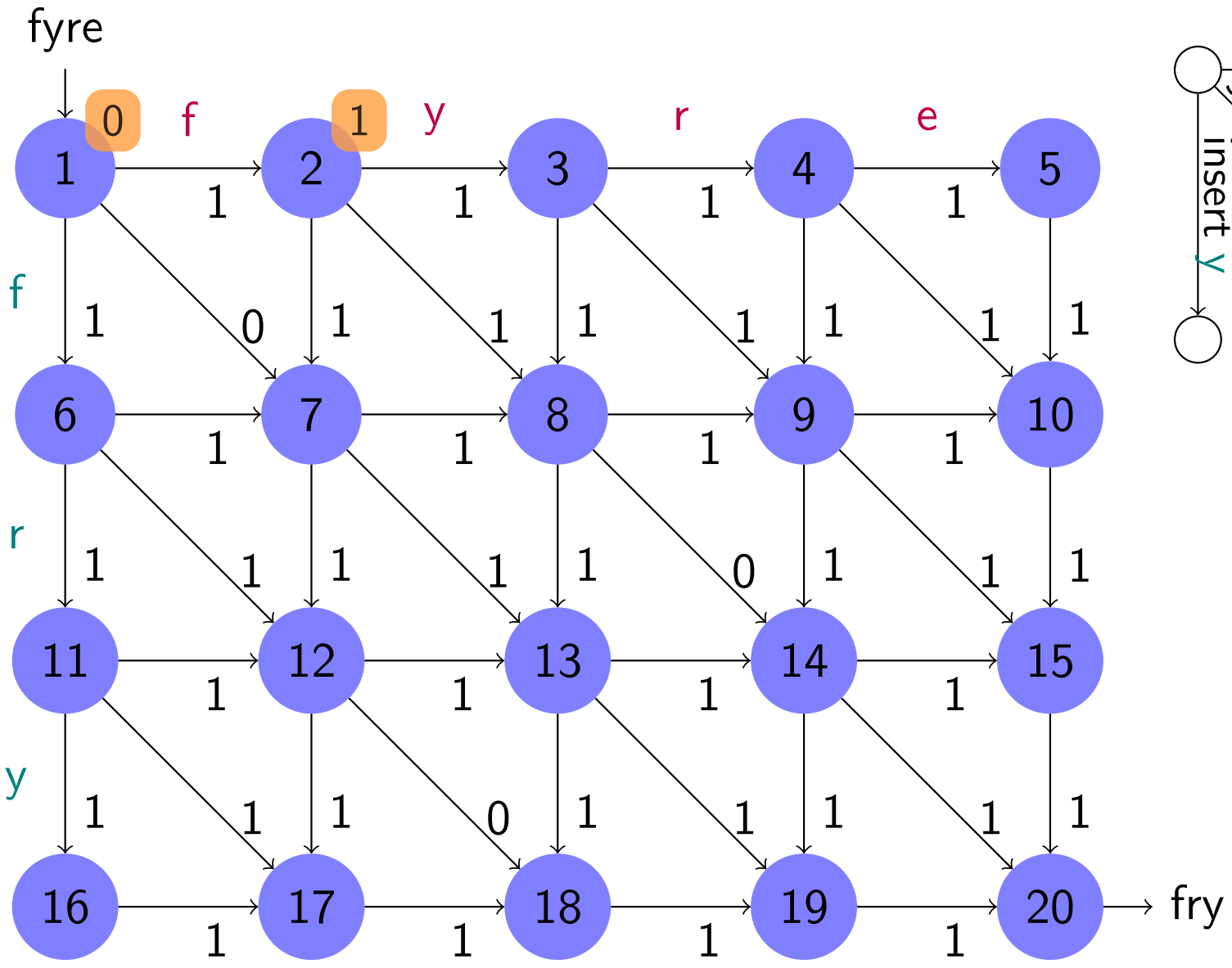


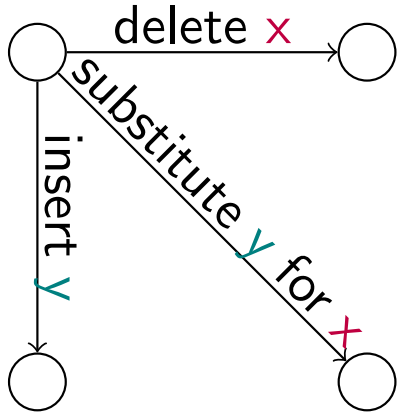
Example of Dynamic Programming



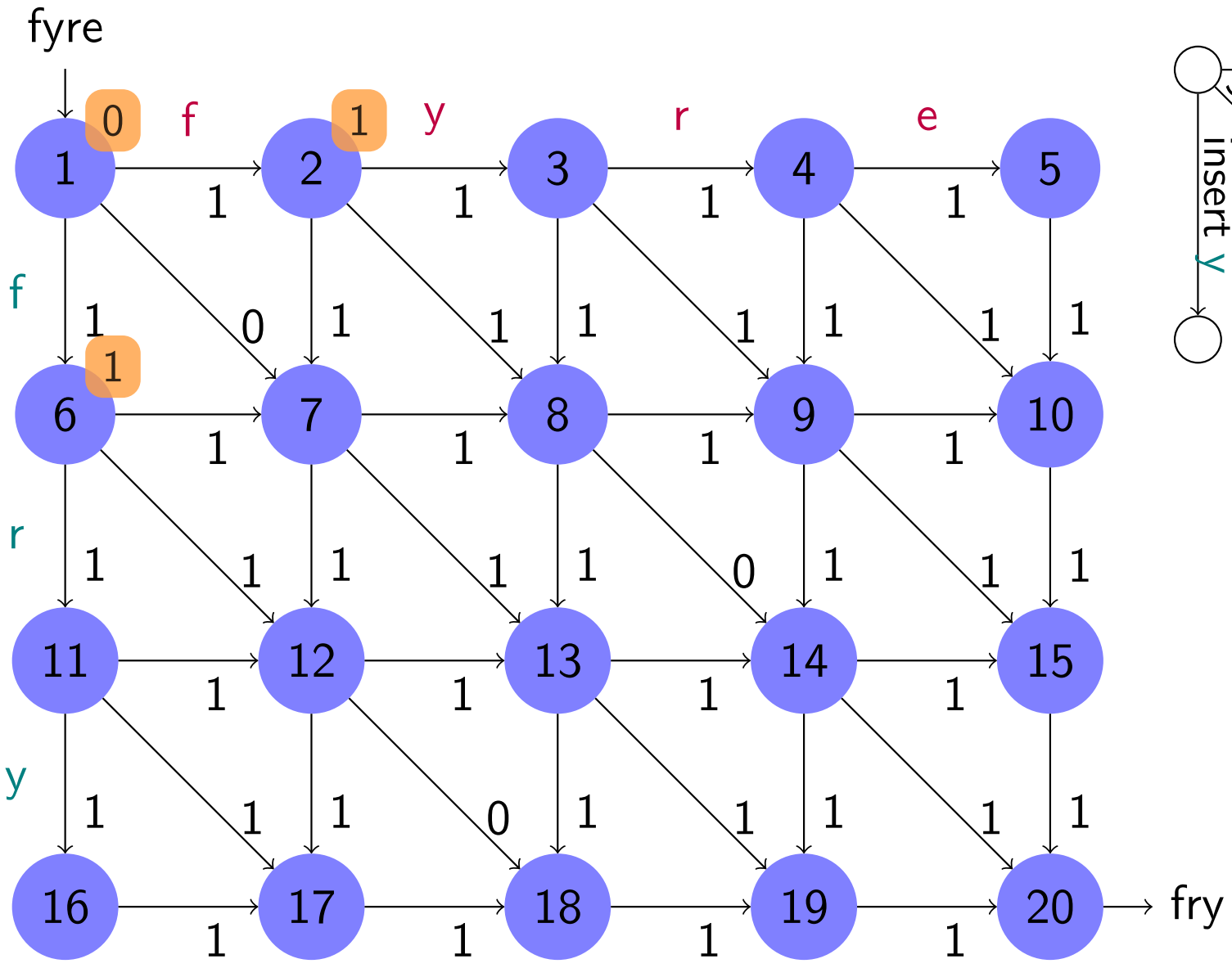


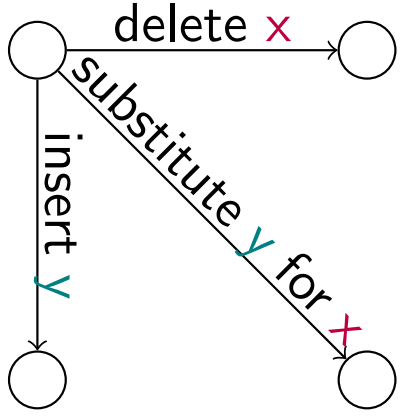
Example of Dynamic Programming



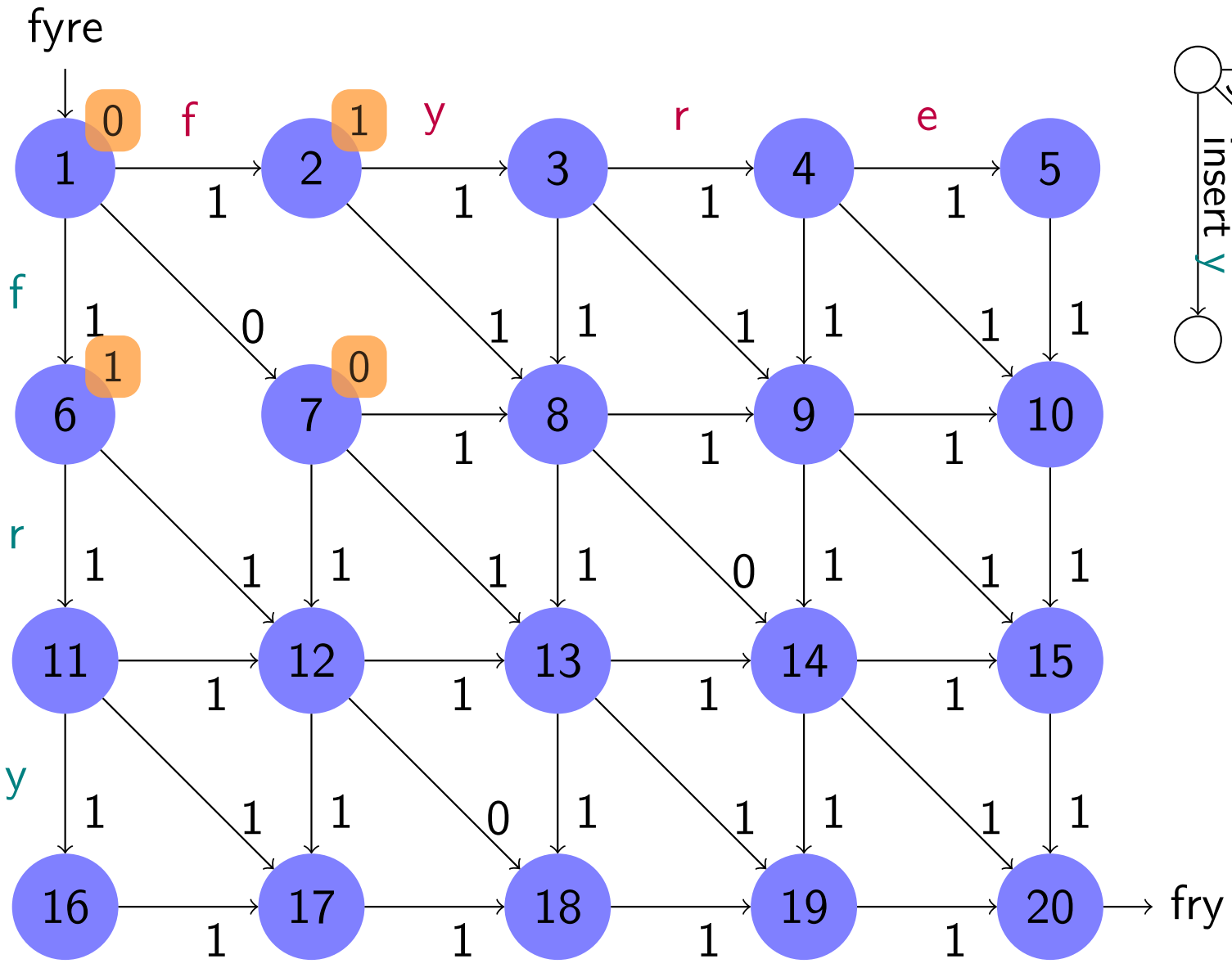


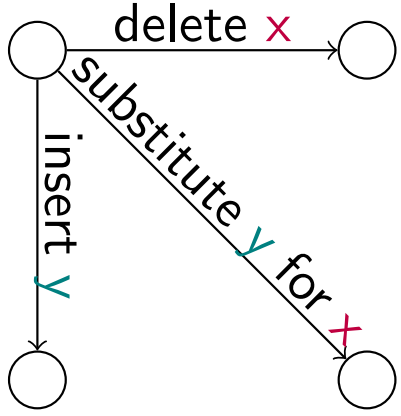
Example of Dynamic Programming



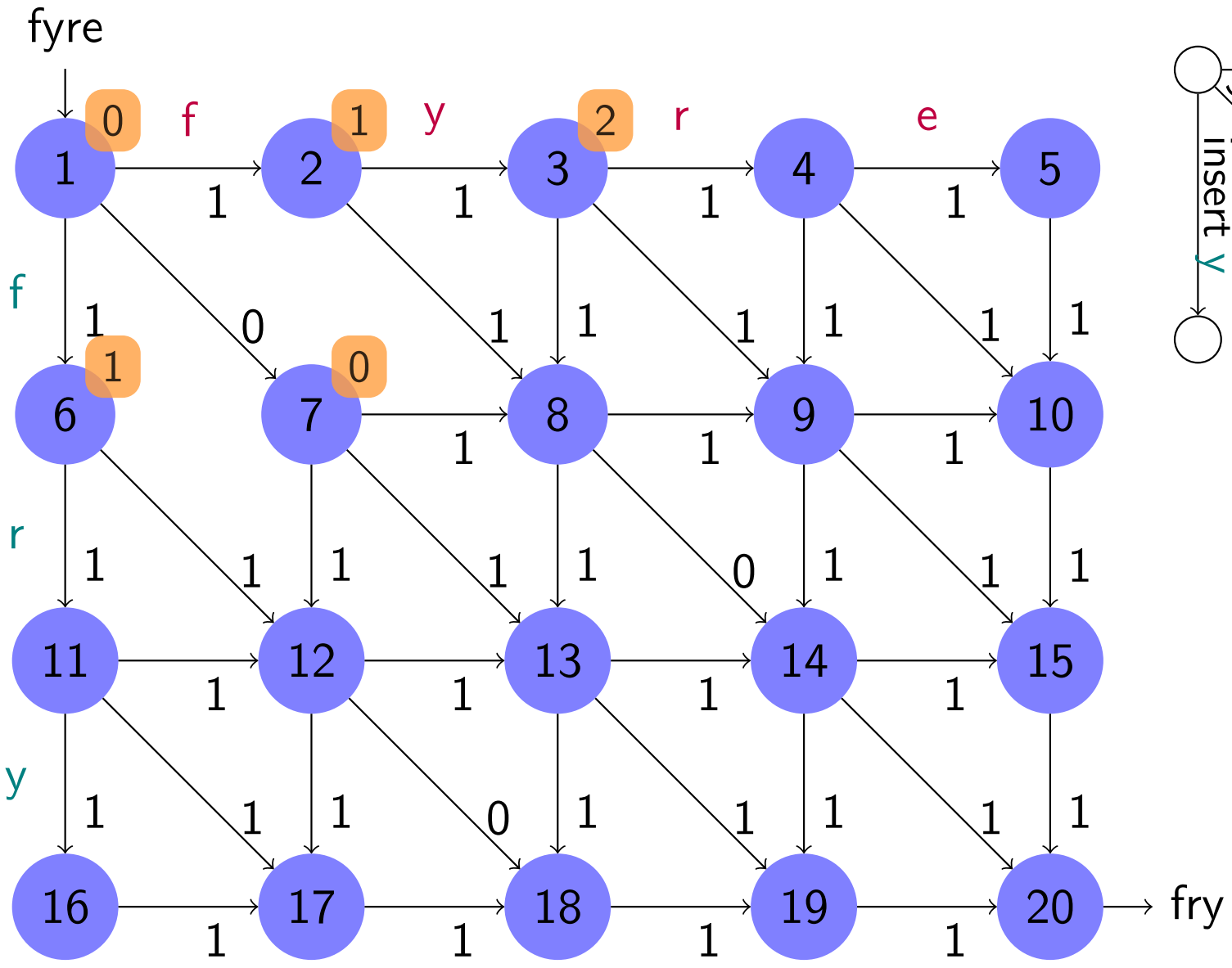


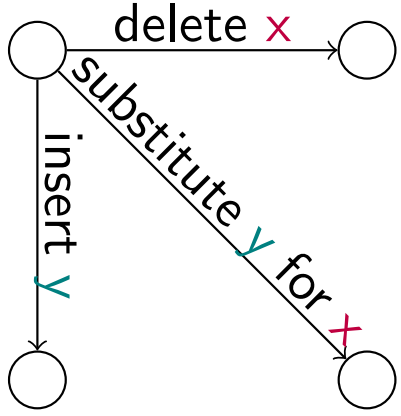
Example of Dynamic Programming



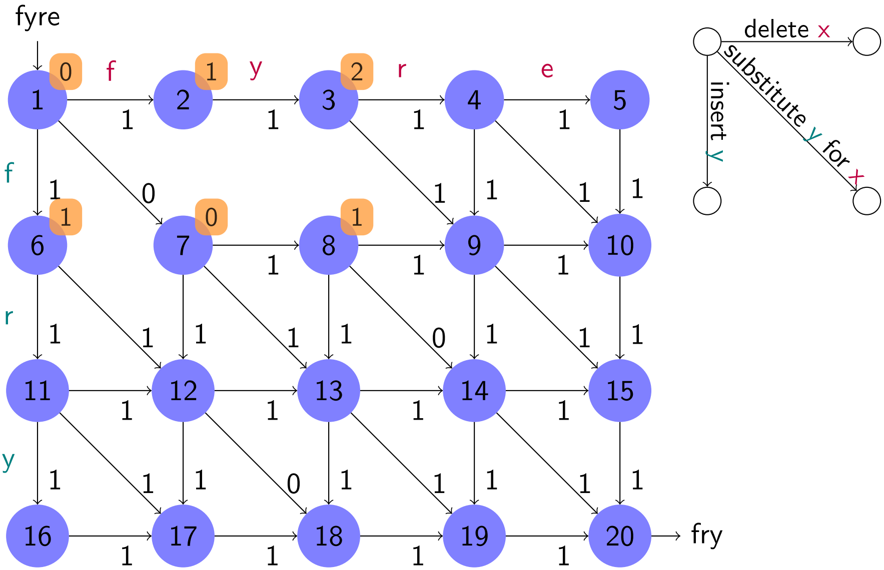


Example of Dynamic Programming

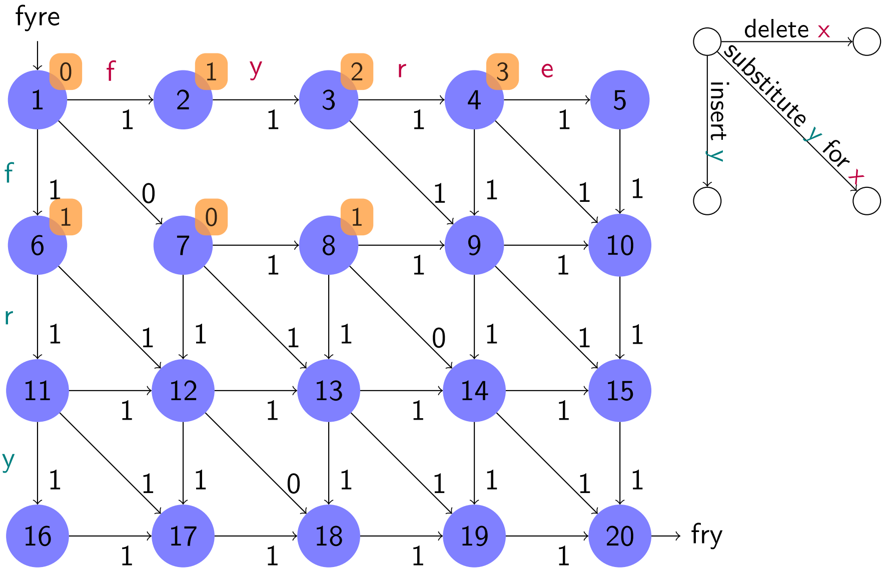




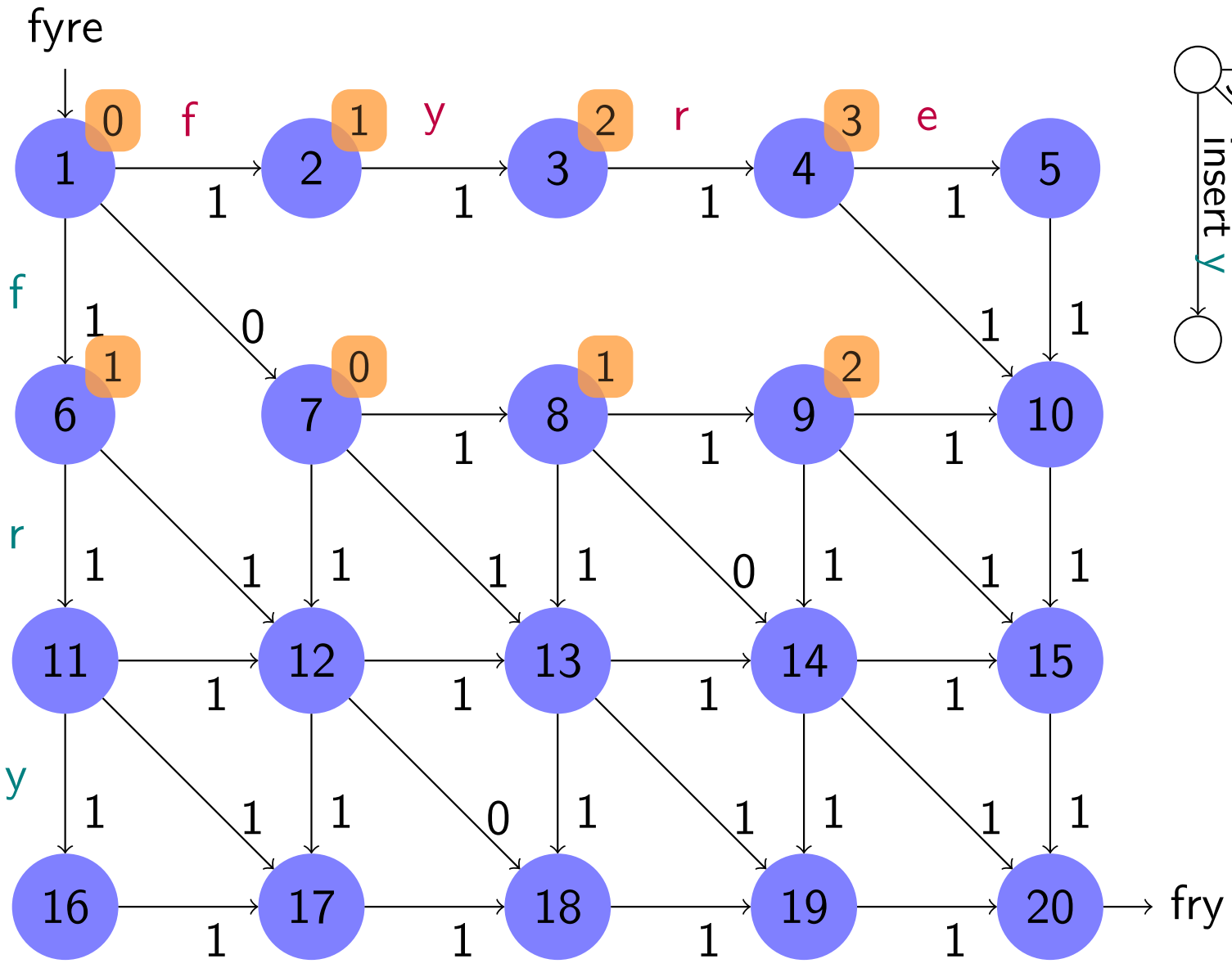
Example of Dynamic Programming



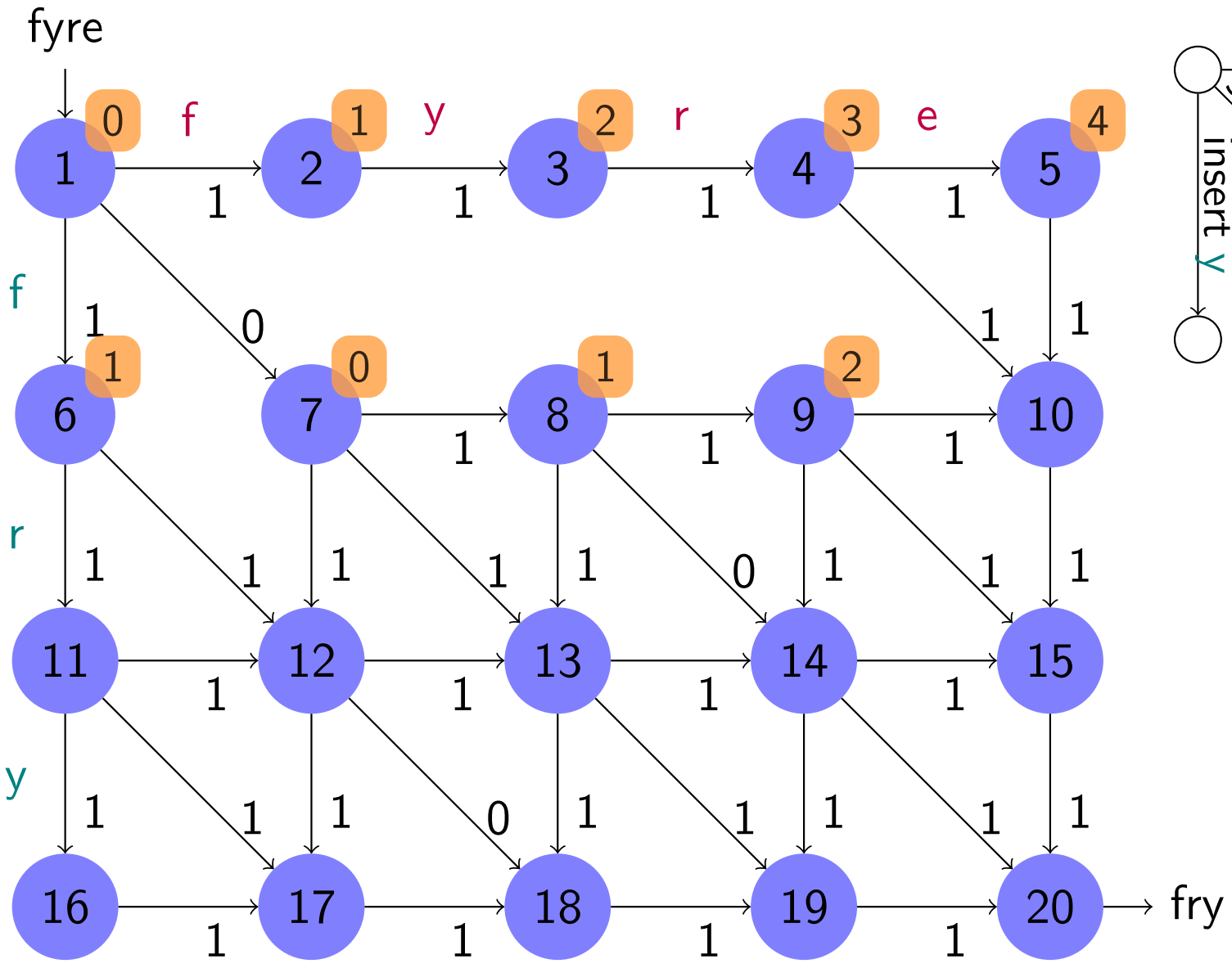
Example of Dynamic Programming

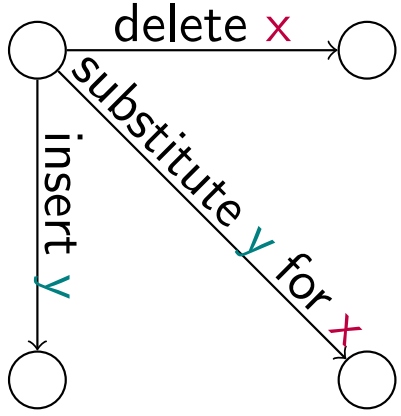


Example of Dynamic Programming

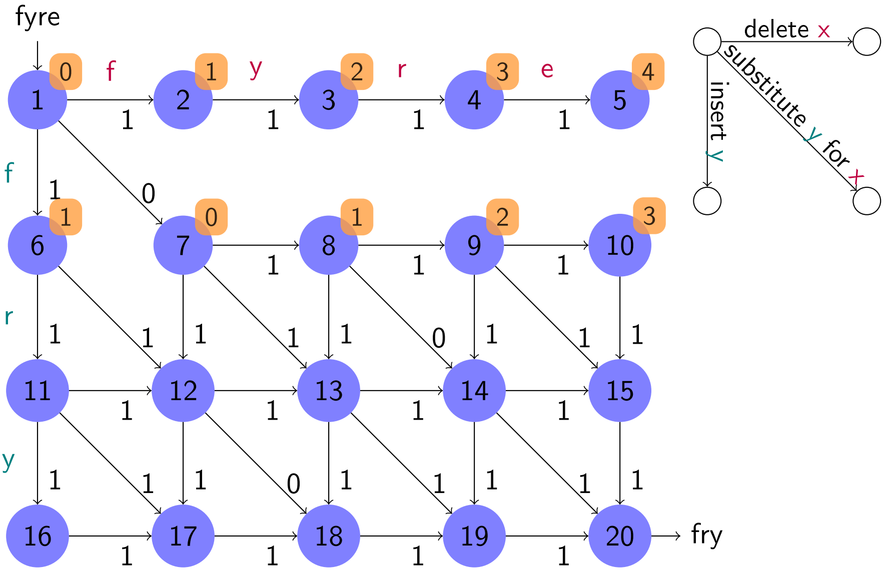


Example of Dynamic Programming

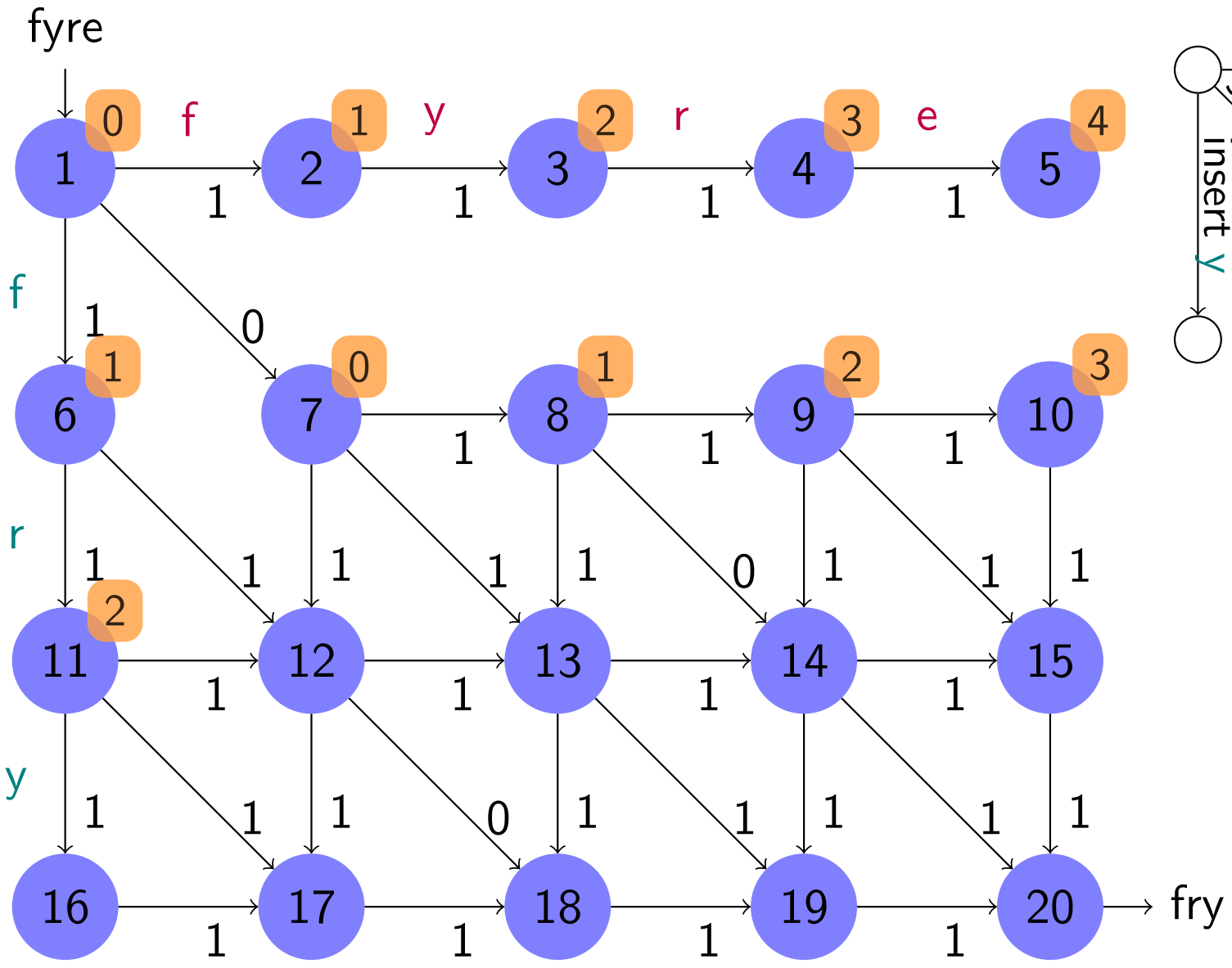




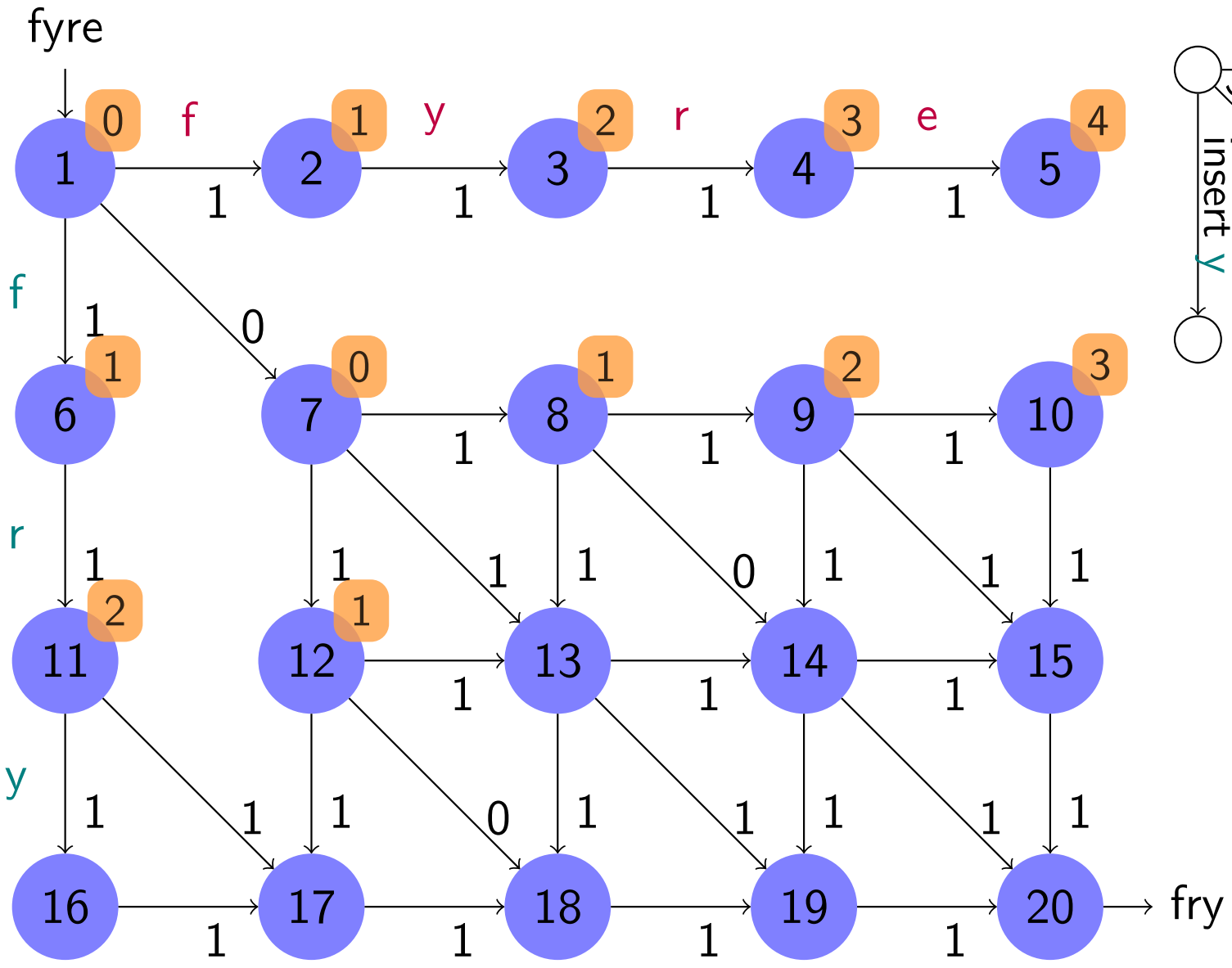
Example of Dynamic Programming



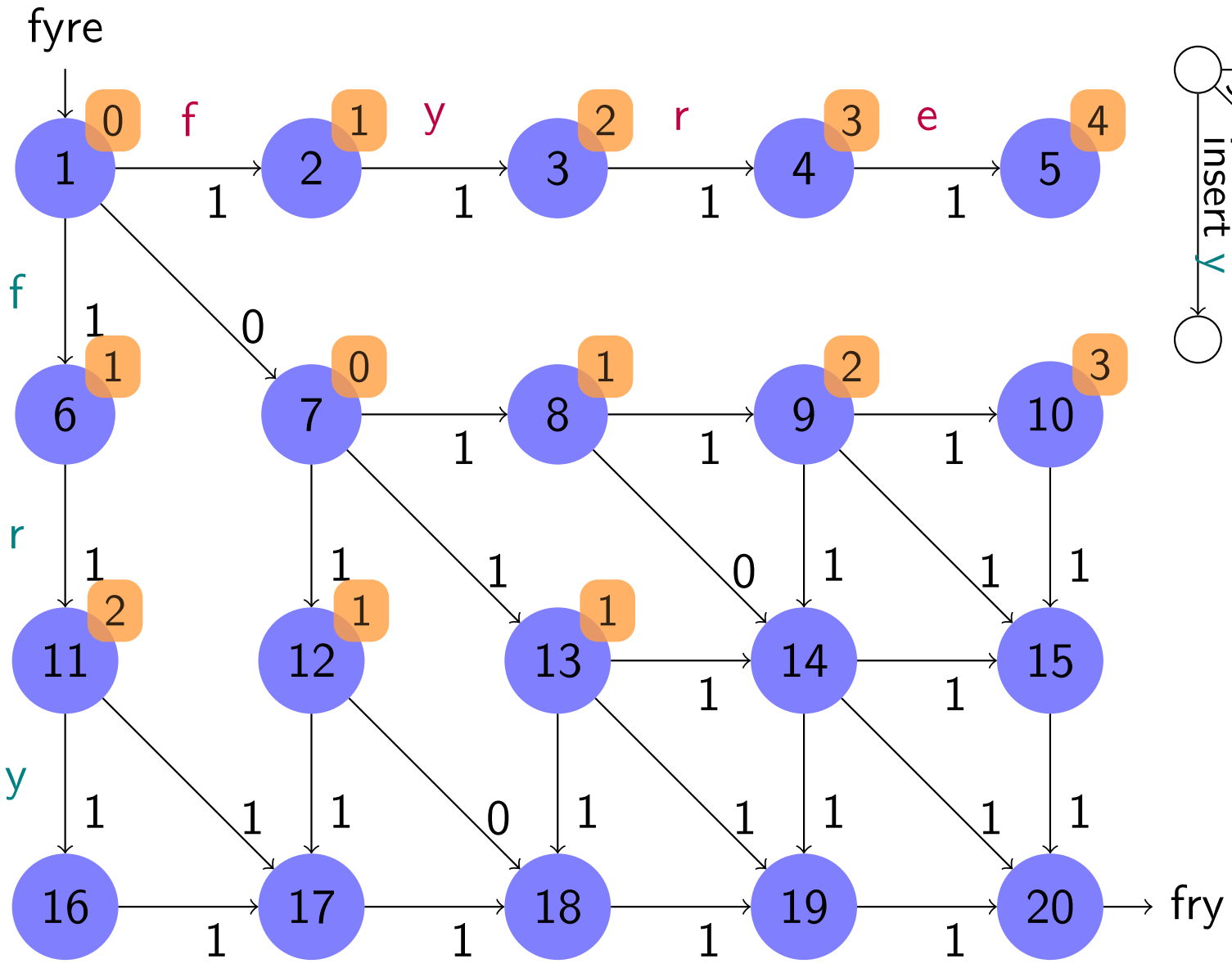
Example of Dynamic Programming



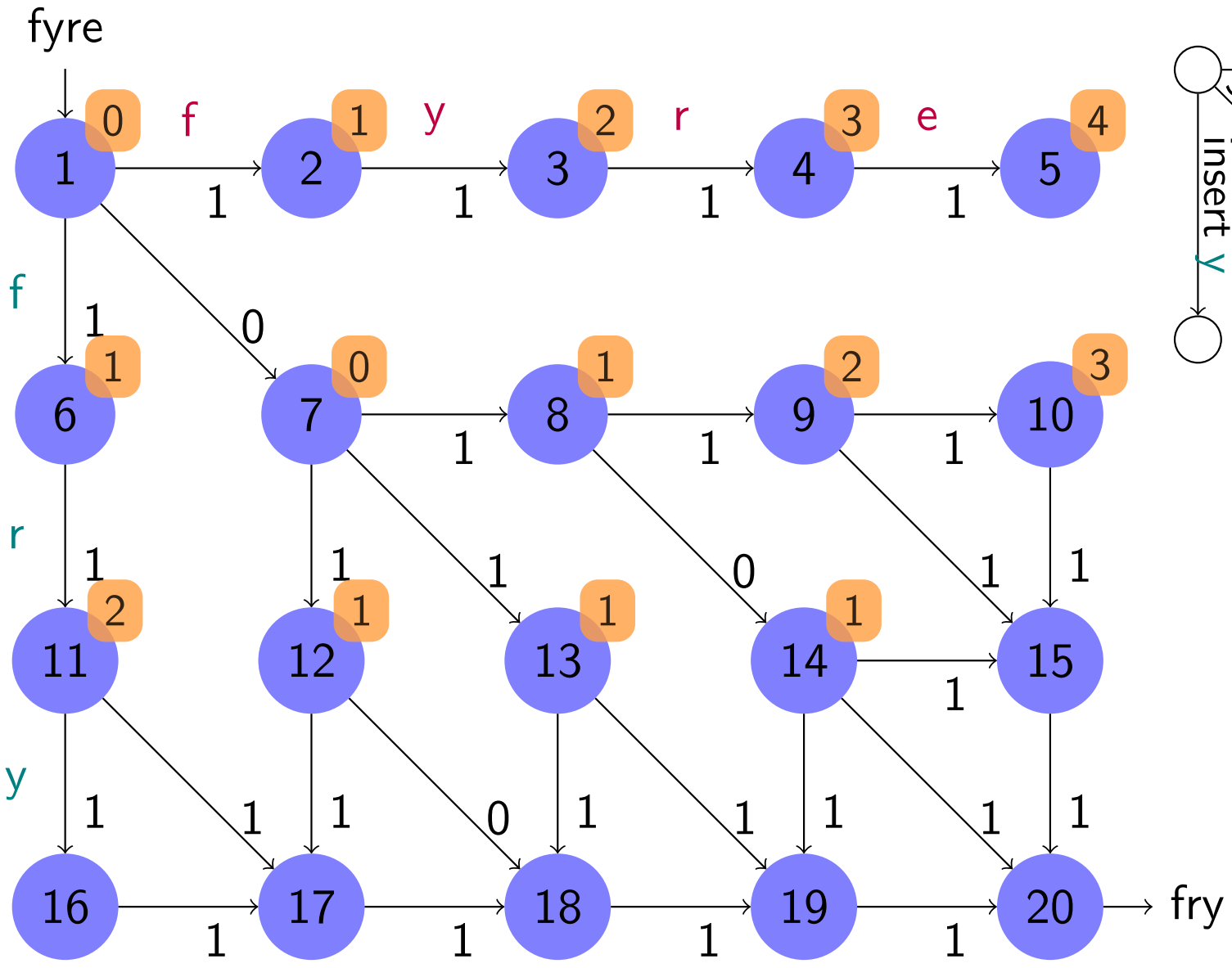
Example of Dynamic Programming



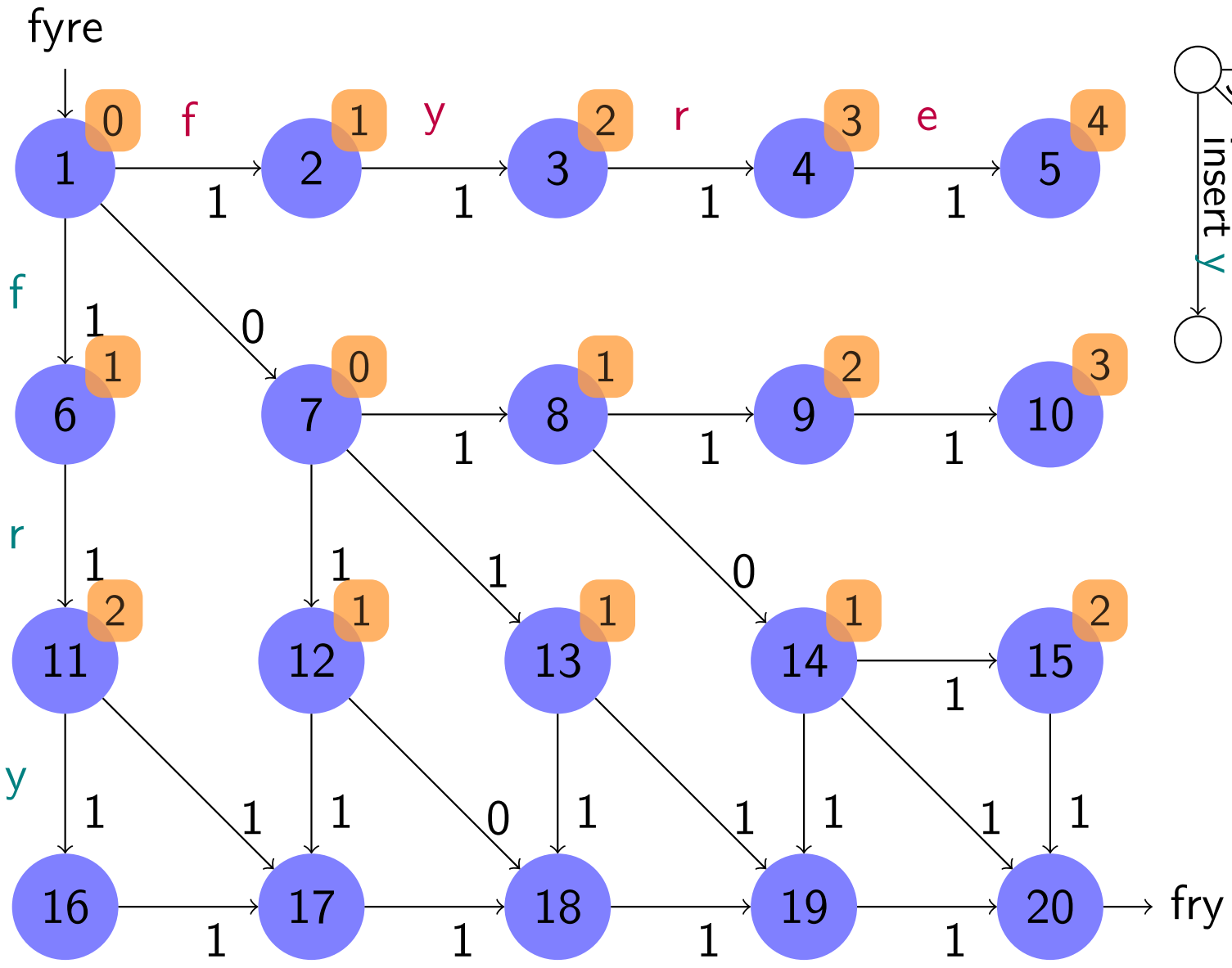
Example of Dynamic Programming



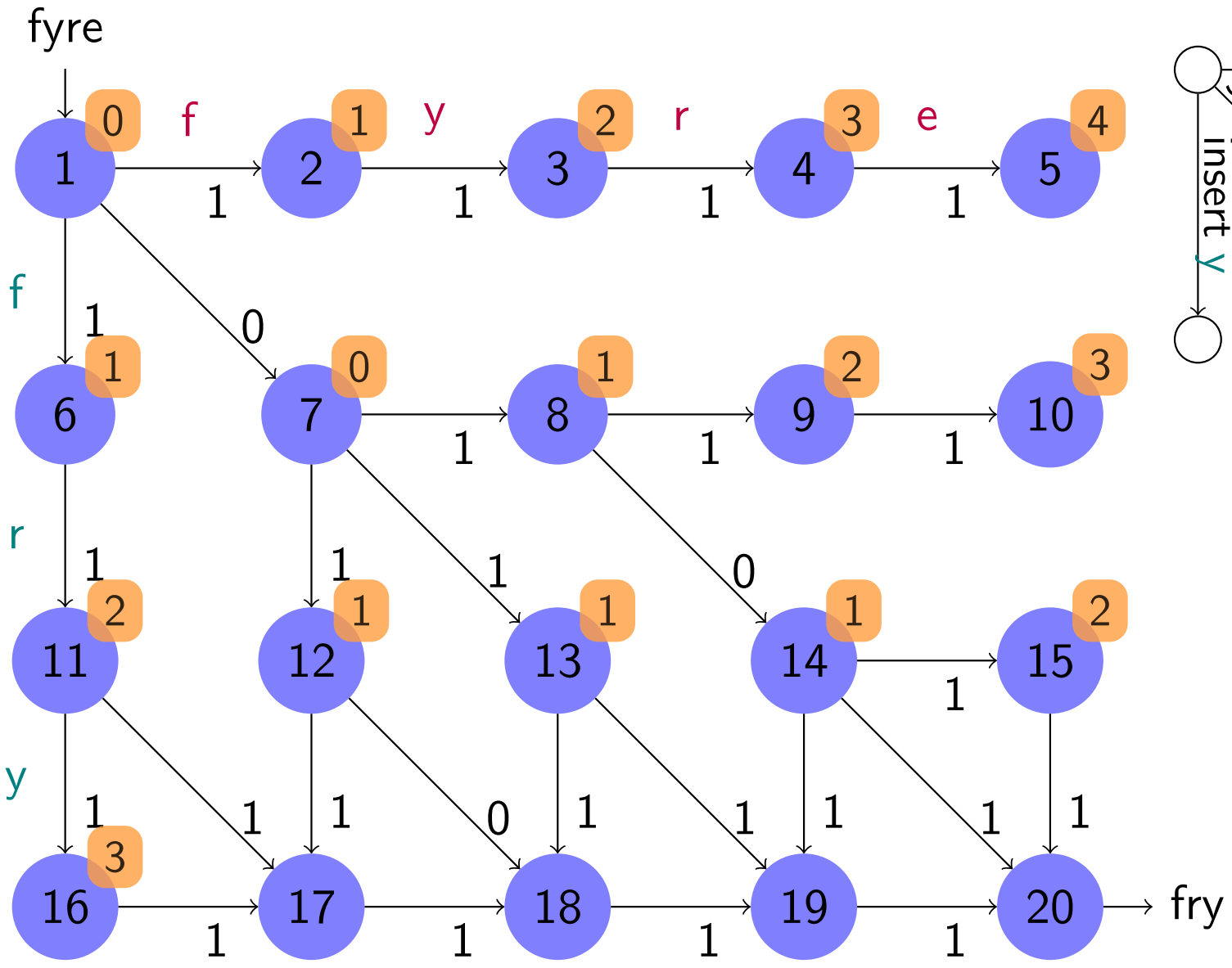
Example of Dynamic Programming



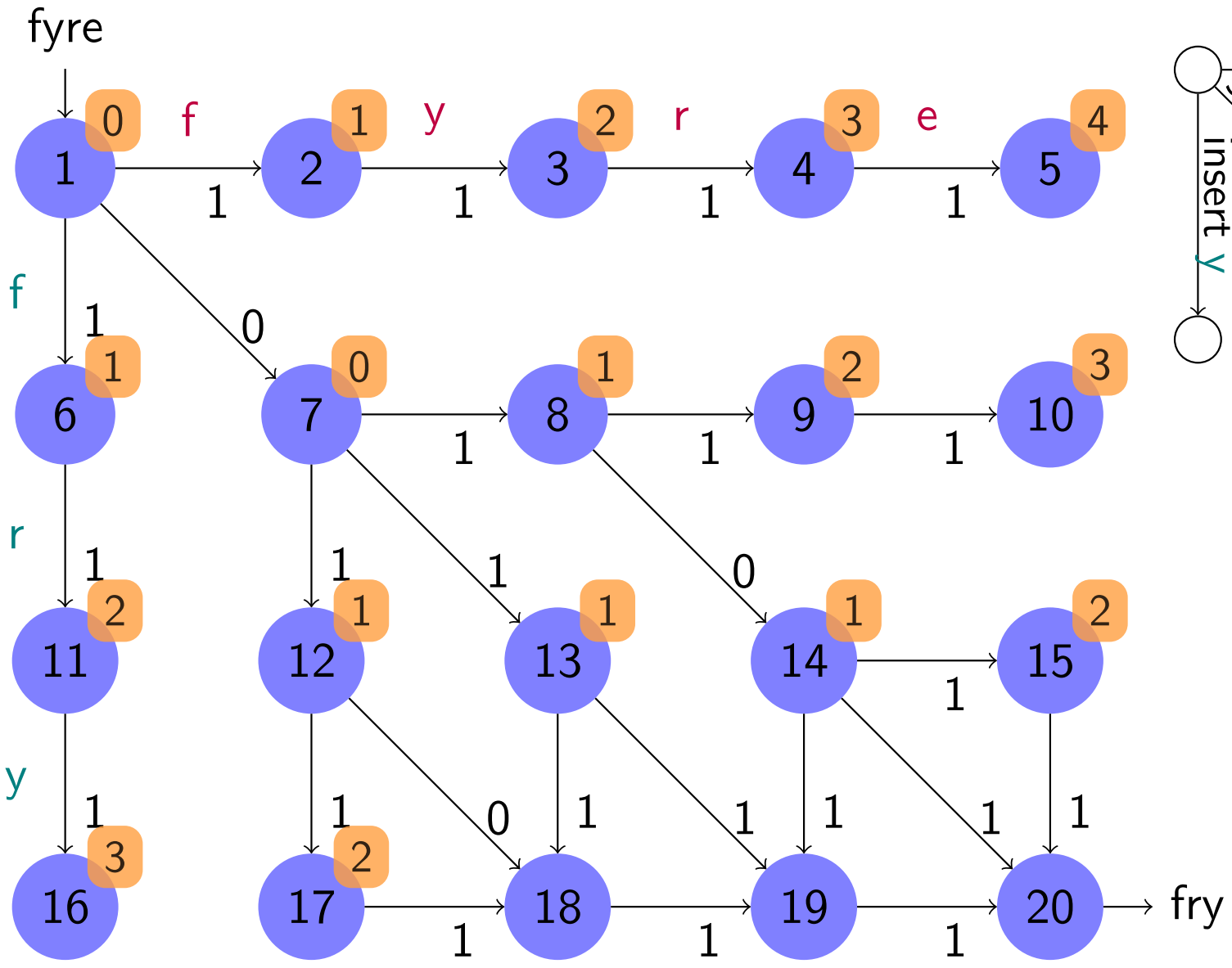
Example of Dynamic Programming



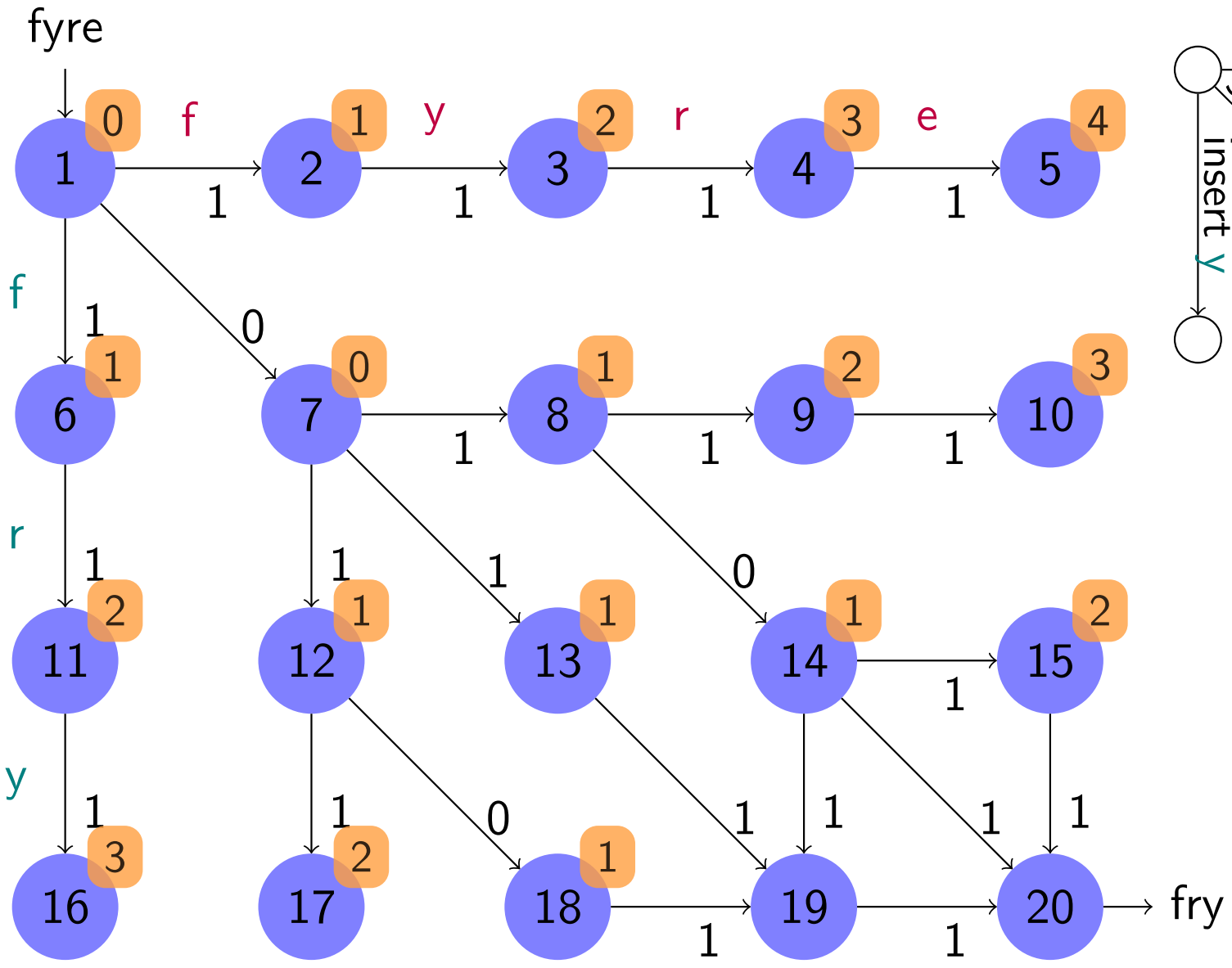
Example of Dynamic Programming



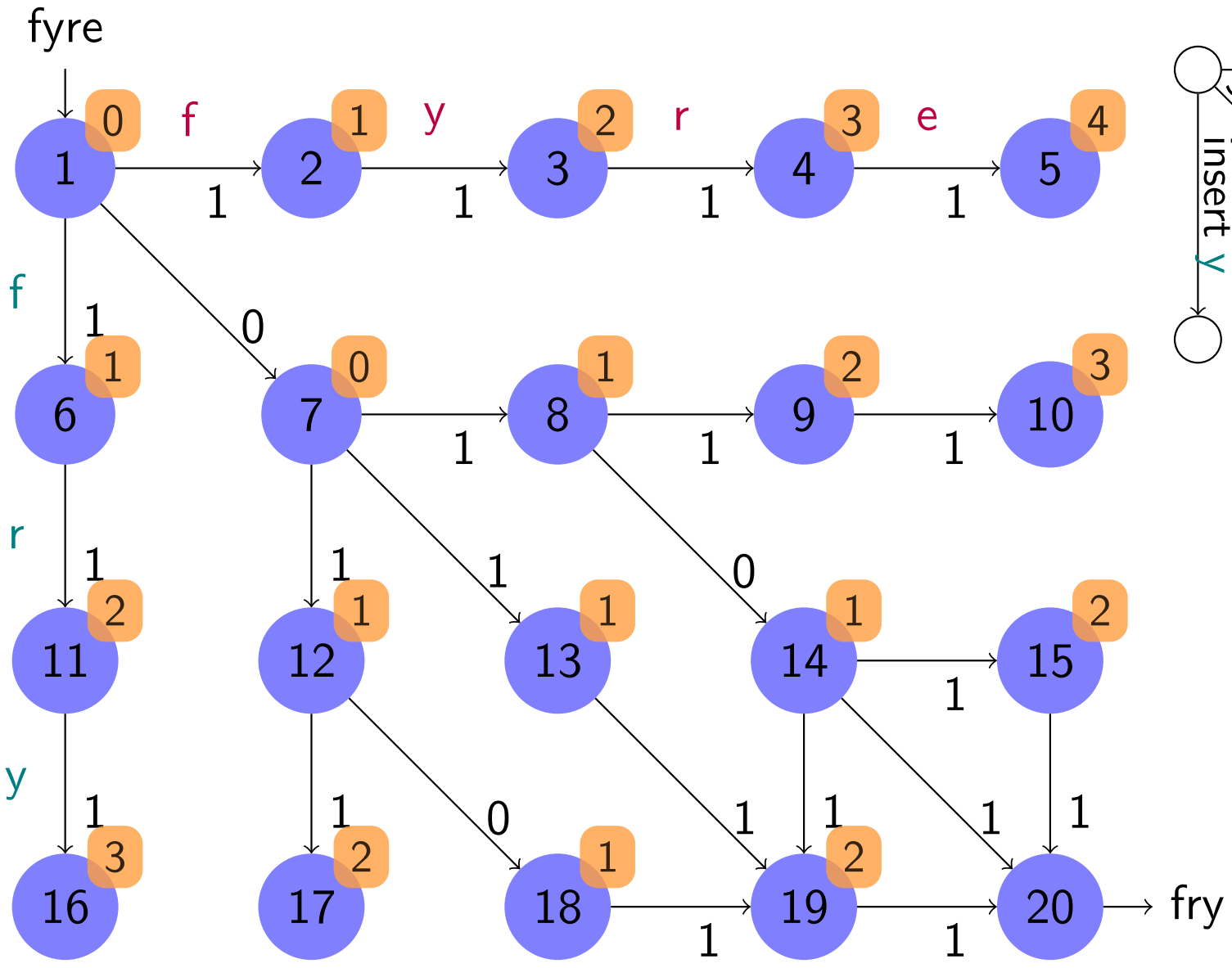
Example of Dynamic Programming



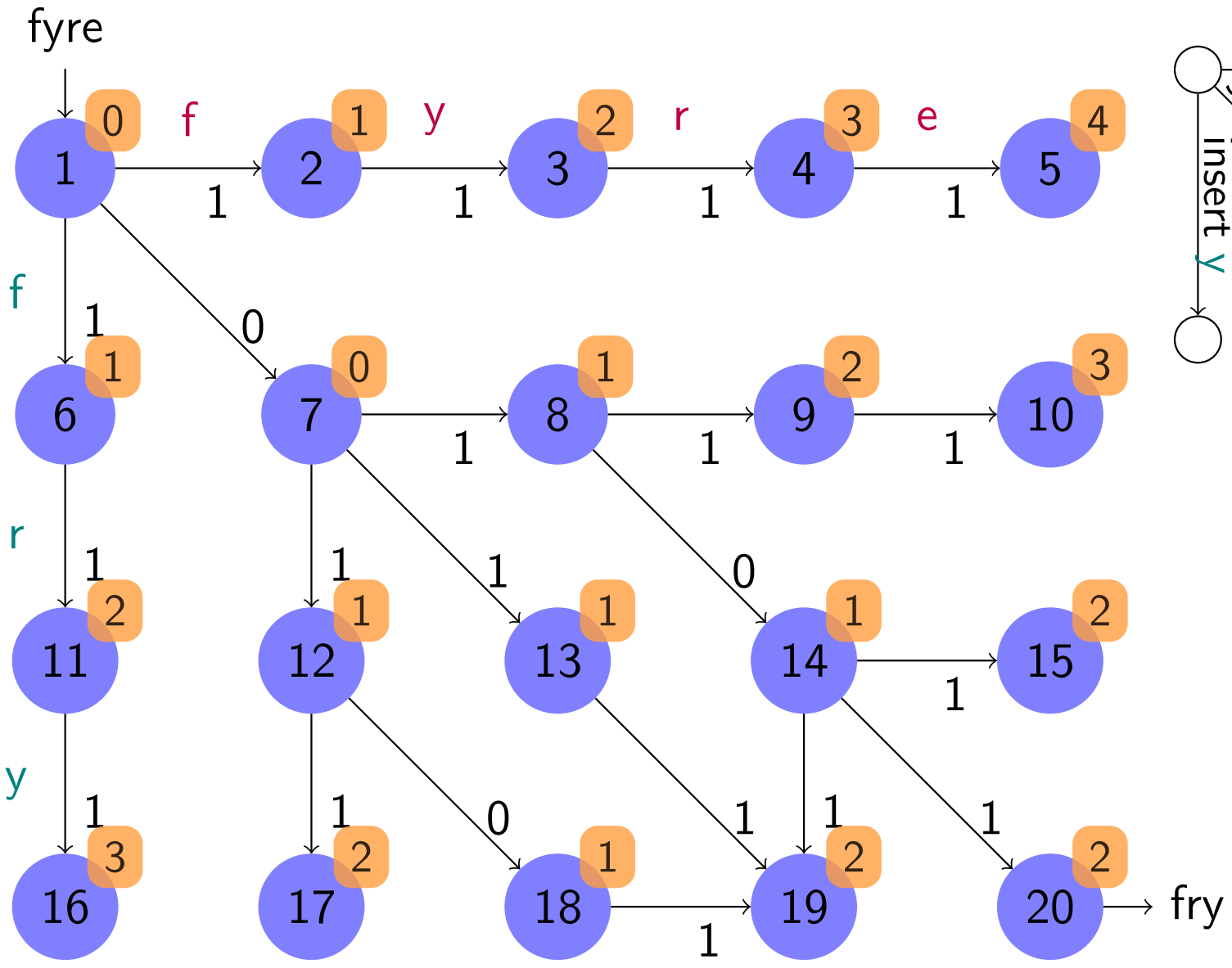
Example of Dynamic Programming



Example of Dynamic Programming



Example of Dynamic Programming



Evaluation of Levenstein Distance

- fully automatic
- more easily applied across languages
- computationally demanding, but
  - dynamic programming tames the beast
  - majority of misspellings have distance ≤ 3

Comparsion of Edit Distance Metrics

Metric	Operations
Damerau-Levenshtein	insert, delete, substitute, transpose
Levenshtein	insert, delete, substitute
longest common subsequence	insert, delete
Hamming	substitute

Evaluation of Levenstein Distance

- fully automatic
- more easily applied across languages
- computationally demanding, but
  - dynamic programming tames the beast
  - majority of misspellings have distance ≤ 3

Comparsion of Edit Distance Metrics

Metric	**Operations**
Damerau-Levenshtein	insert, delete, substitute, transpose
Levenshtein	insert, delete, substitute
longest common subsequence	insert, delete
Hamming	substitute

Adding Probabilities... Again

- Once again probabilities improve accuracy.

Overview of Probability Types

Probability Type	**Application**
corpus frequency	word suggestion in typing
transition probability	improved word suggestion in typing
sentence probability	OCR
word probability	non-word detection
confusion probability	spelling suggestions

- **Confusion probabilities** measure how likely one word is to be typed as realization of another.
- Difficult to compute, affected by many parameters
  keyboard layout, pronunciation, optical similarity, context, ...

Putting it All Together

Step 1: Detecting Spelling Errors

- *Non-Word Errors*
  1. Is the word in the dictionary?
  2. If no, does the word contain illicit character *n*-grams?
  3. If no, is the word an unlikely combination of licit character *n*-grams?
- *Real-Word Errors*
  1. Does the word appear in an illicit *n*-gram?
  2. If no, does the word appear in an unlikely *n*-gram?

Putting it All Together [cont.]

Step 2: Computing and Ranking Possible Corrections

1. *Calculate Set of All Possible Corrections*
   - just use dictionary
   - use naive edit distance algorithm to compute set of possible corrections up to some distance, then remove all words not in dictionary
2. *Ranking Possible Corrections*
   combined heuristic based on
   - Levenshtein distance
   - confusion probability
   - word probability
   - maximizing sentence probability