Dynamic Programming

Georgy Gimel’farb

(with basic contributions by Michael J. Dinneen)

COMPSCI 369 Computational Science

1. Dynamic Programming (DP) Paradigm
2. Discrete Optimisation with DP
3. Viterbi algorithm
4. “0–1” Knapsack Problem: Dynamic programming solution

Learning outcomes:

- Understand DP and problems it can solve
- Be familiar with the edit-distance problem and its DP solution
- Be familiar with the Viterbi algorithm

Additional sources:

- http://en.wikipedia.org/wiki/Dynamic_programming
- http://www.cprogramming.com/tutorial/computersciencetheory/dp.html
- http://en.wikipedia.org/wiki/Knapsack_problem♯Dynamic_programming_solution

Main Algorithmic Paradigms

- **Greedy**: Building up a solution incrementally, by optimising at each step some local criterion
- **Divide-and-conquer**: Breaking up a problem into separate subproblems, solving each subproblem independently, and combining solution to subproblems to form solution to original problem
- **Dynamic programming** (DP): Breaking up a problem into a series of overlapping subproblems, and building up solutions to larger and larger subproblems
  - Unlike the divide-and-conquer paradigm, DP typically involves solving all possible subproblems rather than a small portion
  - DP tends to solve each sub-problem only once, store the results, and use these later again, thus reducing dramatically the amount of computation when the number of repeating subproblems is exponentially large

History of Dynamic Programming

Richard E. Bellman [*26.08.1920 – 19.03.1984*]:

Famous applied mathematician (USA) who pioneered the systematic study of dynamic programming in the 1950s when he was working at RAND Corporation

Etymology:

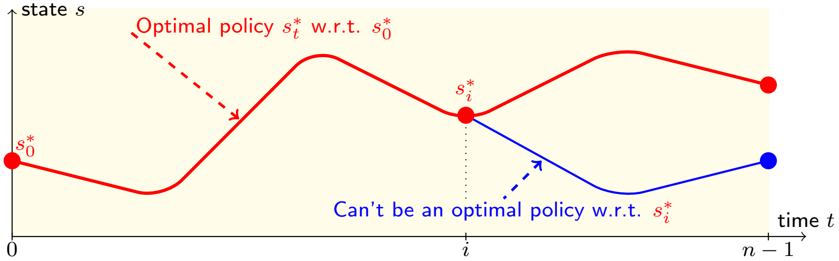
- Dynamic programming = planning over time
- Secretary of Defense was hostile to mathematical research
- Bellman sought an impressive name to avoid confrontation
  - *"It's impossible to use dynamic in a pejorative sense"*
  - *"Something not even a Congressman could object to"*

Reference: Bellman, R. E.: *Eye of the Hurricane*, An Autobiography.

Bellman's Principle of Optimality

R. E. Bellman: *Dynamic Programming*. Princeton Univ. Press, 1957, Ch.III.3

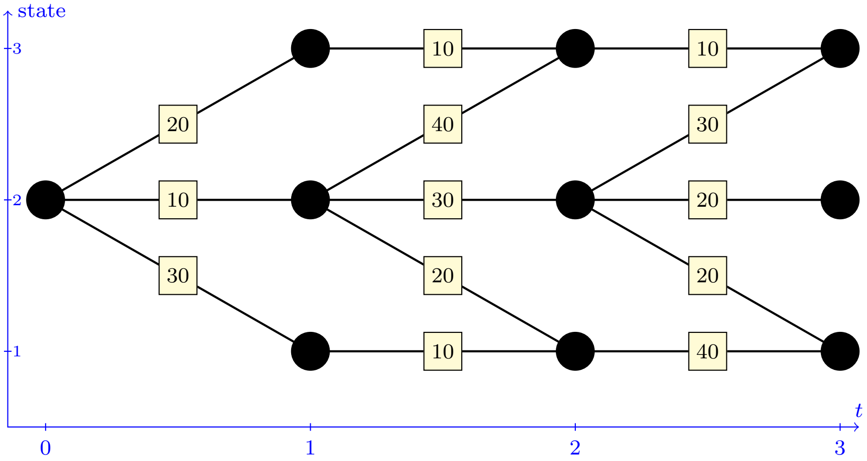
An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision


state s
Optimal policy s_t^* w.r.t. s_0^*
s_i^*
s_0^*
Can't be an optimal policy w.r.t. s_i^*
time t
0
i
$n-1$

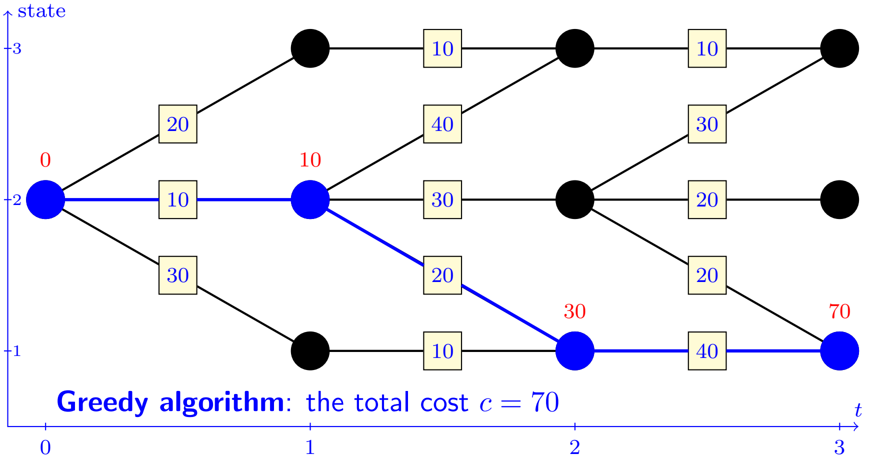
The optimal policy w.r.t. s_0^* after any its state s_i^* cannot differ from the optimal policy w.r.t. the state s_i^*!

See `http://en.wikipedia.org/wiki/Bellman_equation`

Simple Example: Find the Cheapest Route

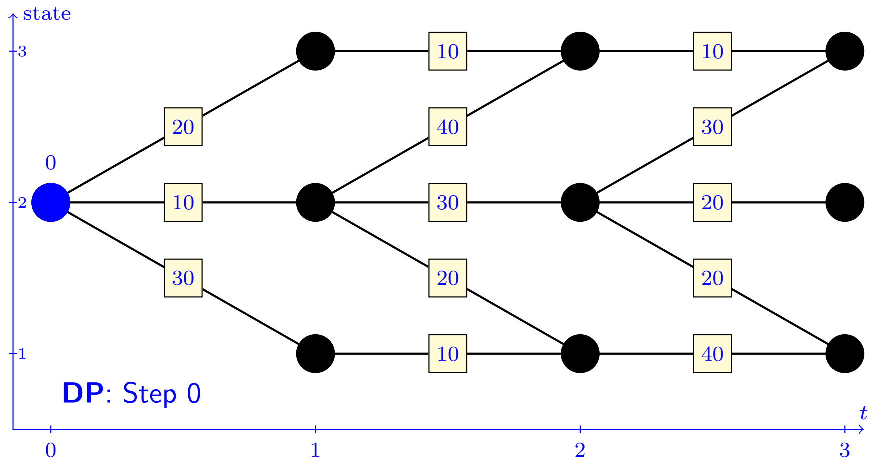


Simple Example: Find the Cheapest Route

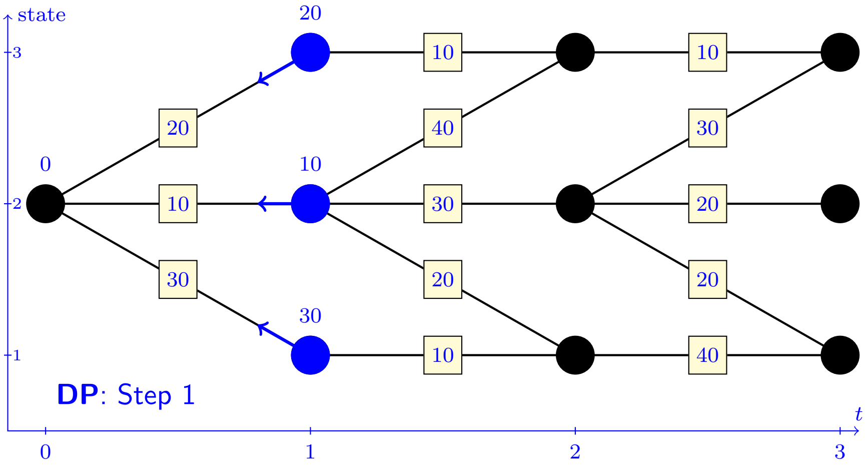


Greedy algorithm: the total cost $c = 70$

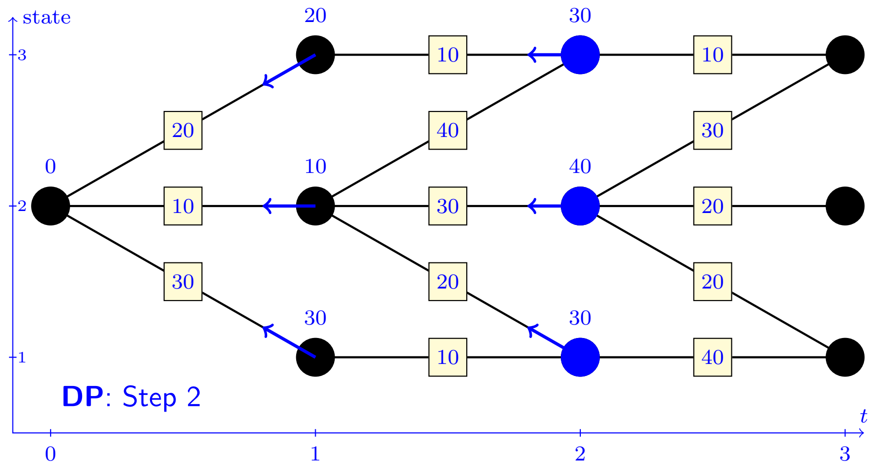
Simple Example: Find the Cheapest Route



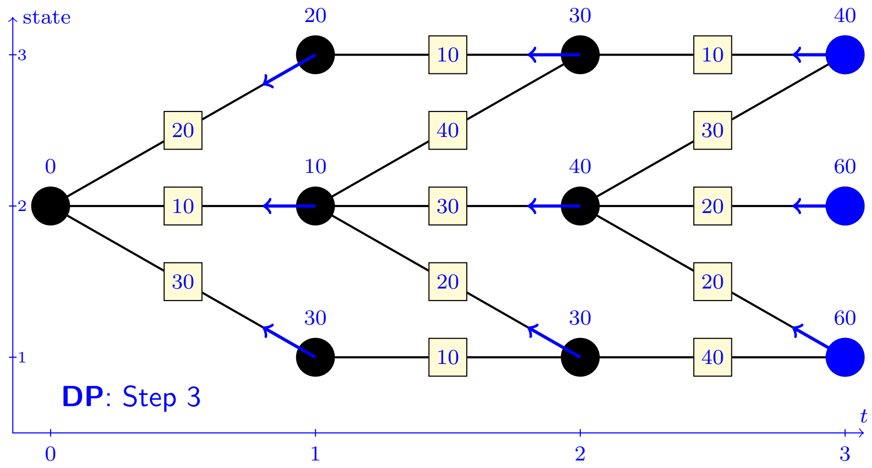
Simple Example: Find the Cheapest Route



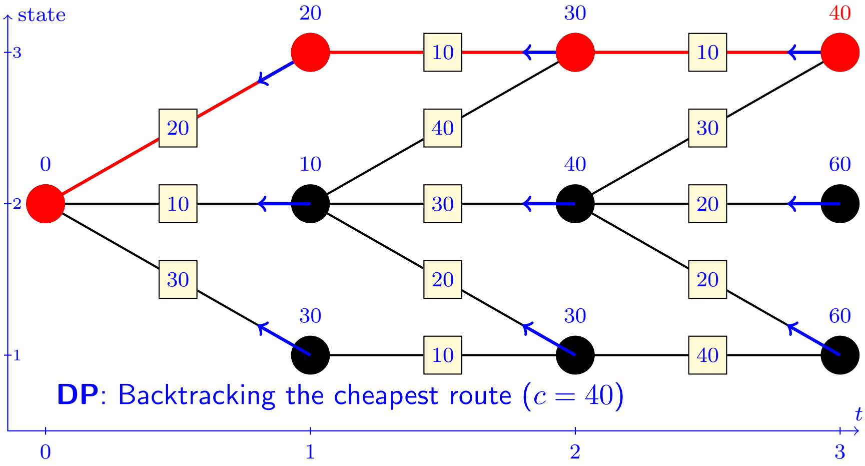
Simple Example: Find the Cheapest Route



Simple Example: Find the Cheapest Route



Simple Example: Find the Cheapest Route



DP: Backtracking the cheapest route ($c = 40$)

Discrete Optimisation with DP

Problem: $(s_0^*, \ldots, s_{n-1}^*) = \arg \min\limits_{(s_i \in \mathbb{S}_i\colon i=0,\ldots,n-1)} F(s_0, \ldots, s_{n-1})$

where an objective function $F(s_0, \ldots, s_{n-1})$ depends on *states* s_i; $i = 0, \ldots, n-1$, having each a finite set $\mathbb{S}_i$ of values

- Frequently, an objective function to apply DP is additive:

$$F(s_0, s_1, \ldots, s_{n-1}) = \psi_o(s_0) + \sum_{i=1}^{n-1} \varphi_i(s_{i-1}, s_i)$$

  - Generally, each state s_i takes only a subset $\mathbf{S}_i(s_{i+1}) \subseteq \mathbb{S}_i$ of values, which depends on the state $s_{i+1} \in \mathbb{S}_{i+1}$
  - Overlapping subproblems are solved for all the states $s_i \in \mathbb{S}_i$ at each step i sequentially for $i = 1, \ldots, n-1$

Computing DP Solution to Problem on Slide 7

Bellman Equation: For $i = 1, \ldots, n-1$ and each $s_i \in \mathbb{S}_i$,

$$\begin{aligned}\Phi_i(s_i) &= \min_{s_{i-1} \in \mathbf{S}_{i-1}(s_i)} \{\Phi_{i-1}(s_{i-1}) + \varphi_i(s_{i-1}, s_i)\} \\ B_i(s_i) &= \arg \min_{s_{i-1} \in \mathbf{S}_{i-1}(s_i)} \{\Phi_{i-1}(s_{i-1}) + \varphi_i(s_{i-1}, s_i)\}\end{aligned}$$

- $\Phi_i(s_i)$ is a candidate decision for state s_i at step i and
- $B_i(s_i)$ is a backward pointer for reconstructing a candidate sequence of states $s_0^\circ, \ldots, s_{i-1}^\circ, s_i$, producing $\Phi_i(s_i)$

Backtracking to reconstruct the solution:

$$\begin{aligned}\min_{s_{n-1} \in \mathbb{S}_{n-1}} \Phi_{n-1}(s_{n-1}) &\equiv \min_{s_0, \ldots, s_{n-1}} F(s_0, \ldots, s_{n-1}) \\ s_{n-1}^* &= \arg \min_{s_{n-1} \in \mathbb{S}_{n-1}} \Phi_{n-1}(s_{n-1}) \\ s_{i-1}^* &= B_i(s_i^*) \text{ for } i = n-1, \ldots, 1\end{aligned}$$

Computing DP Solution to Example on Slide 6

Step i	0	1	2	3	
Set of states $\mathbb{S}_i$	$\{2\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	
State constraints $\mathbf{S}_{i-1}(s_i)$	–	$\{2\}$	$\{1,2\}$	$\{1,2\}$	$s_i = 1$
		$\{2\}$	$\{2\}$	$\{2\}$	$s_i = 2$
		$\{2\}$	$\{2,3\}$	$\{2,3\}$	$s_i = 3$

Cost functions: $\psi_0(2) = 0$

$$\varphi_1(2, s_1) = \left\{ \begin{array}{c|c|} s_1 & \\ \hline 1 & 30 \\ \hline 2 & 10 \\ \hline 3 & 20 \\ \hline \end{array} \right.$$

$$\varphi_2(s_1, s_2) = \left\{ \begin{array}{c|c|c|c|} s_1 \backslash s_2{=}1 & & 2 & 3 \\ \hline 1 & 10 & - & - \\ \hline 2 & 20 & 30 & 40 \\ \hline 3 & - & - & 10 \\ \hline \end{array} \right.$$

$$\varphi_3(s_2, s_3) = \left\{ \begin{array}{c|c|c|c|} s_2 \backslash s_3{=}1 & & 2 & 3 \\ \hline 1 & 40 & - & - \\ \hline 2 & 20 & 20 & 30 \\ \hline 3 & - & - & 10 \\ \hline \end{array} \right.$$

Step $i = 0$: $\Phi_0(s_0 = 2) = \psi_0(2) = 0$

Step $i = 1$: $\Phi_1(s_1) =$

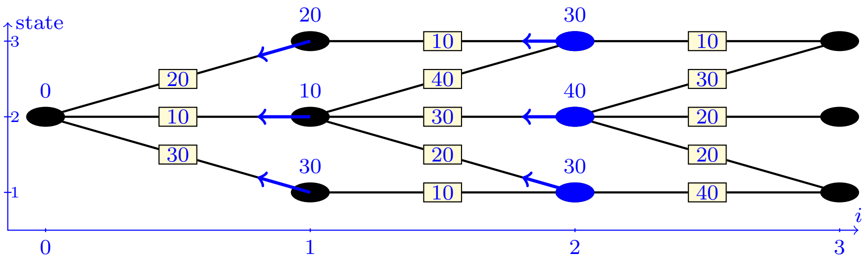
$$\left\{ \begin{array}{llcl} s_1 = 1 & \rightarrow & \underbrace{\Phi_0(2)}_{0} + \underbrace{\varphi_1(2,1)}_{30} = & 30 \\ s_1 = 2 & \rightarrow & \underbrace{\Phi_0(2)}_{0} + \underbrace{\varphi_1(2,2)}_{10} = & 10 \\ s_1 = 3 & \rightarrow & \underbrace{\Phi_0(2)}_{0} + \underbrace{\varphi_1(2,3)}_{20} = & 20 \end{array} \right.$$

$B_1(s_1) = 2$ for $s_1 = 1, 2, 3$

Computing DP Solution to Example on Slide 6 (continued)

Step $i = 2$:

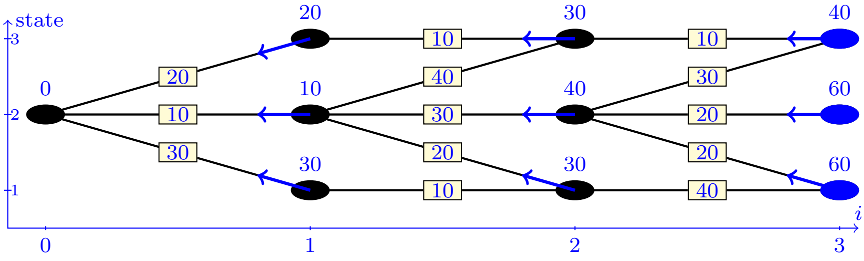
$$\Phi_2(s_2) = \begin{cases} s_2 = 1 & \rightarrow & \min\{\underbrace{\Phi_1(1) + \varphi_2(1,1)}_{30+10}, \underbrace{\Phi_1(2) + \varphi_2(1,2)}_{10+20}\} = 30; & B_2(1) = 2 \\ s_2 = 2 & \rightarrow & \underbrace{\Phi_1(2) + \varphi_2(2,2)}_{10+30} = 40; & B_2(2) = 2 \\ s_2 = 3 & \rightarrow & \min\{\underbrace{\Phi_1(2) + \varphi_2(2,3)}_{10+40}, \underbrace{\Phi_1(2) + \varphi_2(3,3)}_{20+10}\} = 30; & B_2(3) = 3 \end{cases}$$



Computing DP Solution to Example on Slide 6 (continued)

Step $i = 3$:

$$\Phi_3(s_3) = \begin{cases} s_3 = 1 & \rightarrow & \min\{\underbrace{\Phi_2(1) + \varphi_3(1,1)}_{30+40}, \underbrace{\Phi_2(2) + \varphi_3(1,2)}_{40+20}\} = 60; & B_3(1) = 2 \\ s_3 = 2 & \rightarrow & \underbrace{\Phi_2(2) + \varphi_2(2,2)}_{40+20} = 60; & B_3(2) = 2 \\ s_3 = 3 & \rightarrow & \min\{\underbrace{\Phi_2(2) + \varphi_2(2,3)}_{40+30}, \underbrace{\Phi_2(3) + \varphi_3(3,3)}_{20+10}\} = 30; & B_3(3) = 3 \end{cases}$$

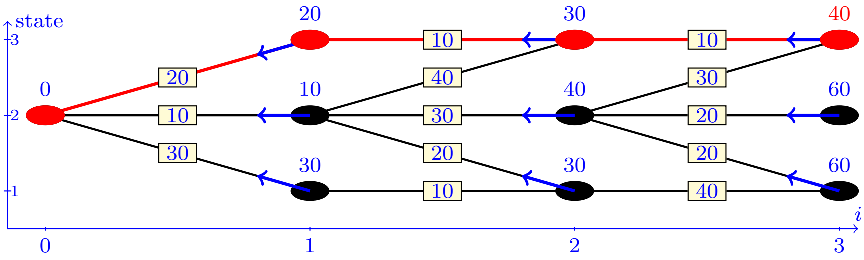


Computing DP Solution to Example on Slide 6 (continued)

Solution:

- Optimal solution: $\min\limits_{s_0,\ldots,s_3} F(s_0,\ldots,s_3) \equiv \min\limits_{s_3\in\mathbb{S}_3} \Phi_3(s_3) = \min\{60,60,40\} = 40$
- Ending optimal state: $s_3^* = \min\limits_{s_3\in\mathbb{S}_3} \Phi_3(s_3) = 3$
- Backtracking preceding states for the optimal solution:

$$s_2^* = B_3(3) = 3 \;\rightarrow\; s_1^* = B_2(3) = 3 \;\rightarrow\; s_0^* = B_1(3) = 2$$



DP: Applications and Algorithms

Areas:

- Control theory
- Signal processing
- Information theory
- Operations research
- Bioinformatics
- Computer science: theory, AI, graphics, image analysis, systems, . . .

Algorithms

- Viterbi: error correction coding, hidden Markov models
- Unix `diff`: comparing two files
- Smith-Waterman: gene sequence alignment
- Bellman-Ford: shortest path routing in networks
- Cocke-Kasami-Younger: parsing context free grammars

Levenshtein, or Edit Distance: Minimum Cost of Editing

Applications:
Unix `diff`, speech recognition, computational biology

Different penalties for insertion – deletion, $\delta > 0$, and for mismatch between two characters x and y: $\alpha_{xy} \geq 0$; $\alpha_{xx} = 0$

c	l	a	i	m	–
–	l	–	i	m	e
δ	$\alpha_{\text{ll}} = 0$	δ	$\alpha_{\text{ii}} = 0$	$\alpha_{\text{mm}} = 0$	δ

$\text{cost}(\text{claim} \rightarrow \text{lime}) = 3\delta$

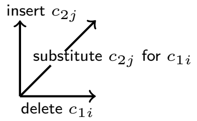
c	l	a	i	m	–
l	–	–	i	m	e
α_{cl}	δ	δ	$\alpha_{\text{ii}} = 0$	$\alpha_{\text{mm}} = 0$	δ

$\text{cost}(\text{claim} \rightarrow \text{lime}) = \alpha_{\text{cl}} + 3\delta$

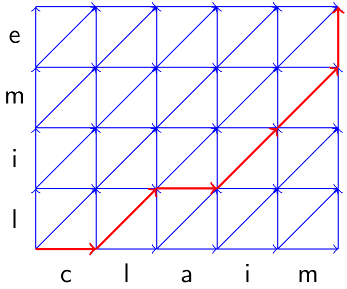
http://en.wikipedia.org/wiki/Edit_distance

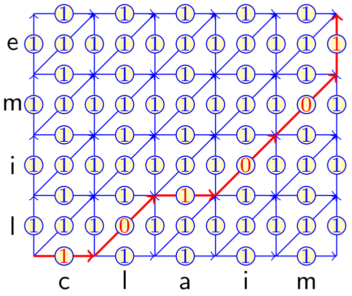
Two Strings C_1 and C_2: Levenshtein, or Edit Distance

Minimum number $D(C_1, C_2)$ of edit operations to transform C_1 into C_2: insertion (weight δ), deletion (δ), or character substitution (0 if the same character, otherwise $\alpha_{..} > 0$); e.g. $\delta = \alpha_{..} = 1$



$D(\text{claim}, \text{lime}) = 3 \Leftrightarrow$	1)	**claim**	$\rightarrow$	laim
	2)	**l**aim	$\rightarrow$	lim
	3)	lim	$\rightarrow$	lim**e**

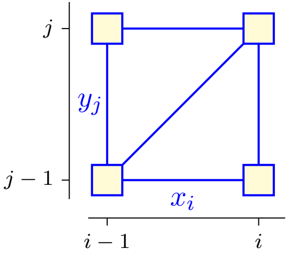




http://en.wikipedia.org/wiki/Levenshtein_distance

Levenshtein, or Edit Distance: DP Computation

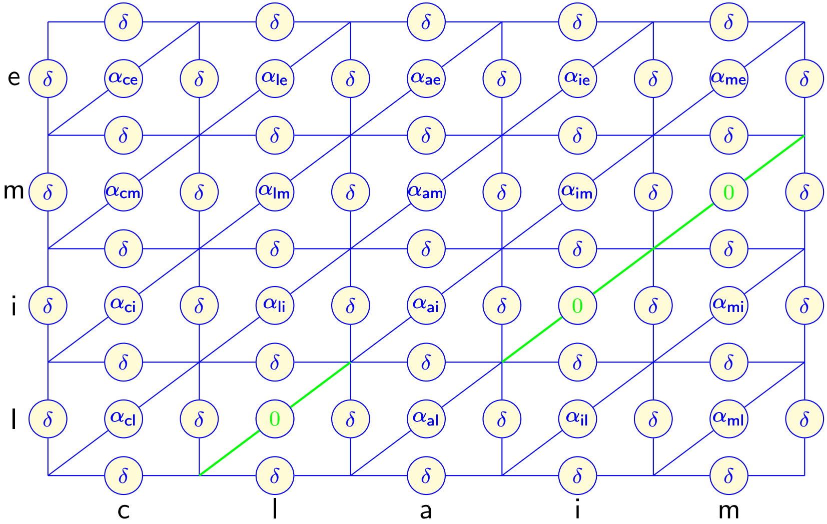
- Strings: $x \equiv x_{[m]} = x_1 x_2 \dots x_m$ and $y \equiv y_{[n]} = y_1 y_2 \dots y_n$
- Substrings: $x_{[i]} = x_1 \dots x_i$; $1 \le i \le m$, and $y_{j]} = y_1 \dots y_j$; $1 \le j \le n$
- Distance $d(i,j) = D(x_{[i]}, y_{[j]})$



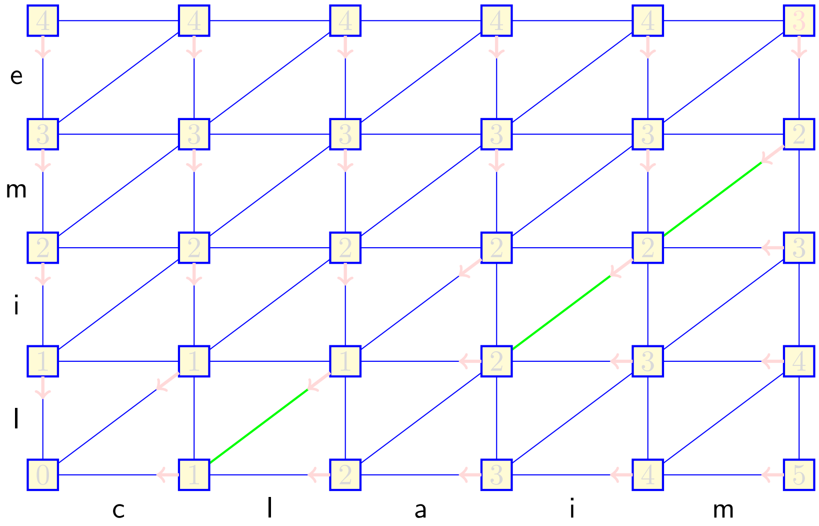
Recurrent computation:

$$d(i,j) = \begin{cases} 0 & \text{if } i=0; j=0 \\ i \equiv d(i-1,0)+\delta & \text{if } i>0; j=0 \\ j \equiv d(0,j-1)+\delta & \text{if } i=0; j>0 \\ \min \left\{ \begin{array}{l} d(i-1,j)+\delta, \\ d(i,j-1)+\delta, \\ d(i-1,j-1)+\underbrace{\alpha_{x_i y_j}}_{\ge 0;\ \alpha_{xx}=0} \end{array} \right\} & \text{otherwise} \end{cases}$$

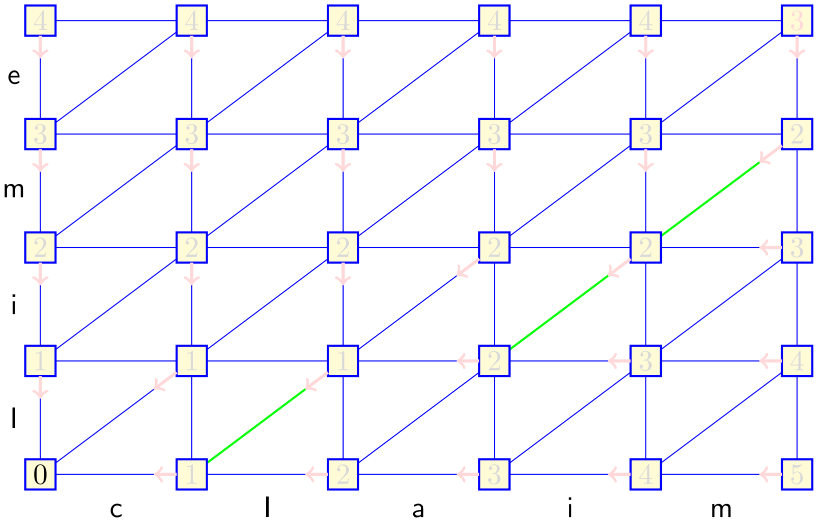
Levenshtein, or Edit Distance: DP Computation



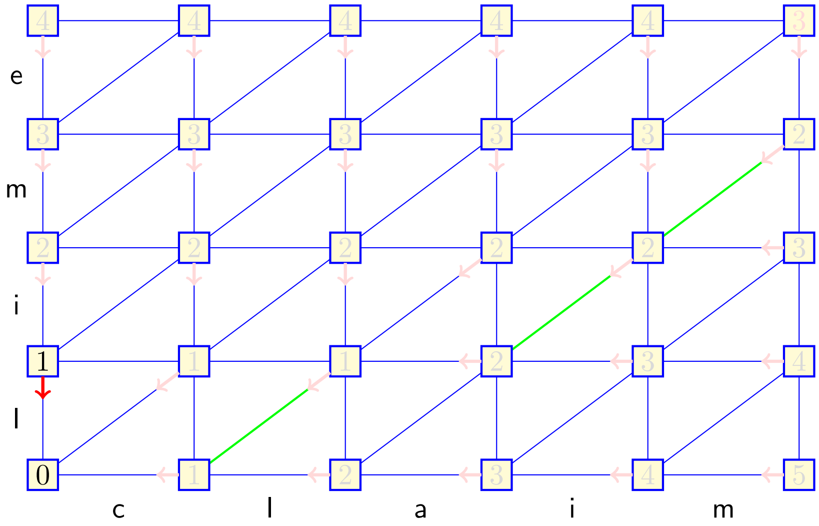
Levenshtein, or Edit Distance: DP Computation



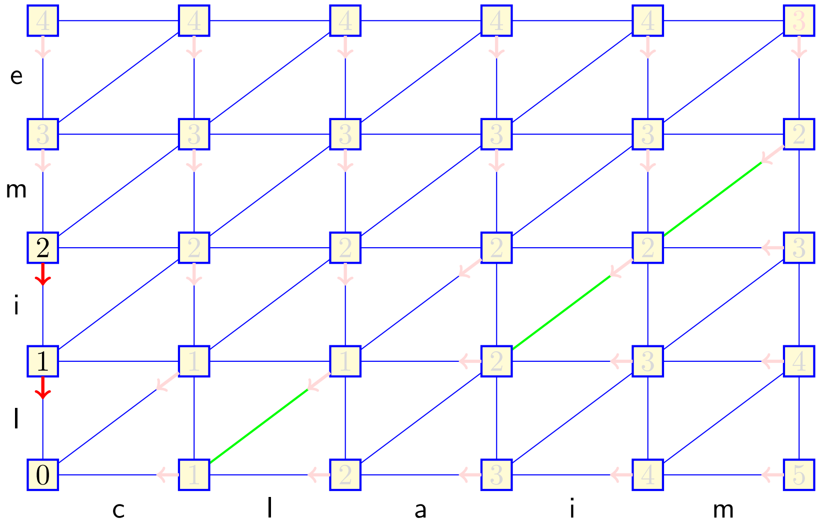
Levenshtein, or Edit Distance: DP Computation



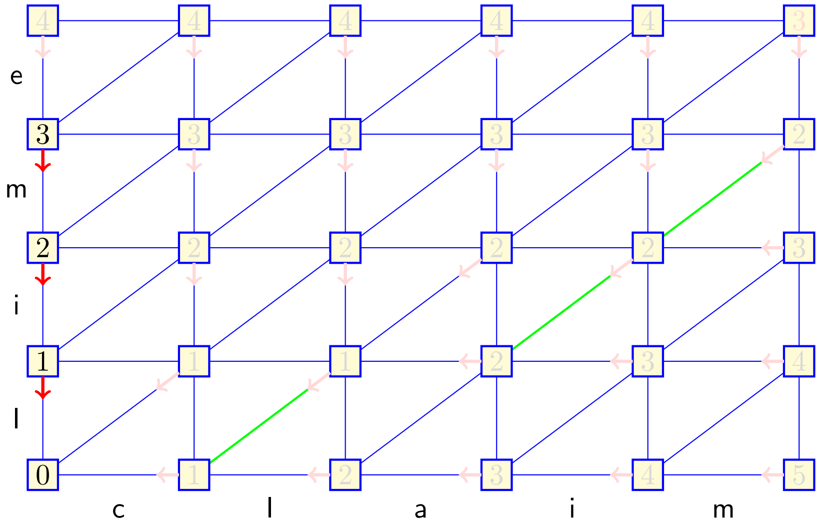
Levenshtein, or Edit Distance: DP Computation



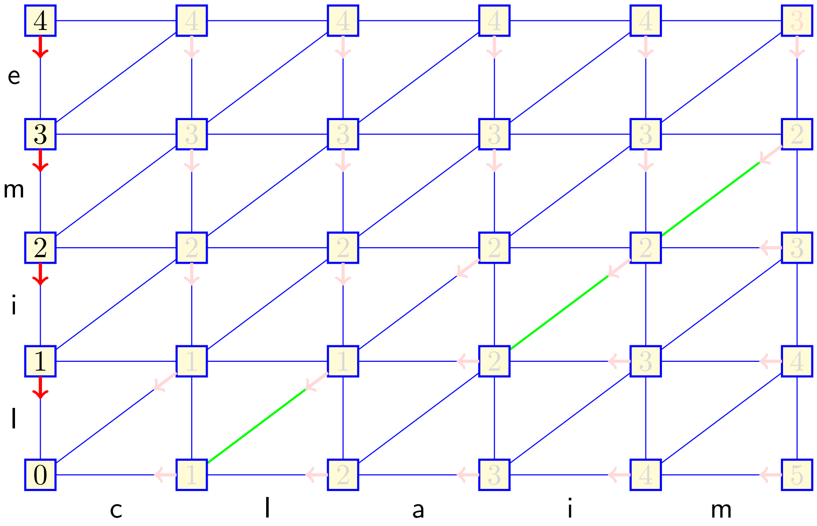
Levenshtein, or Edit Distance: DP Computation



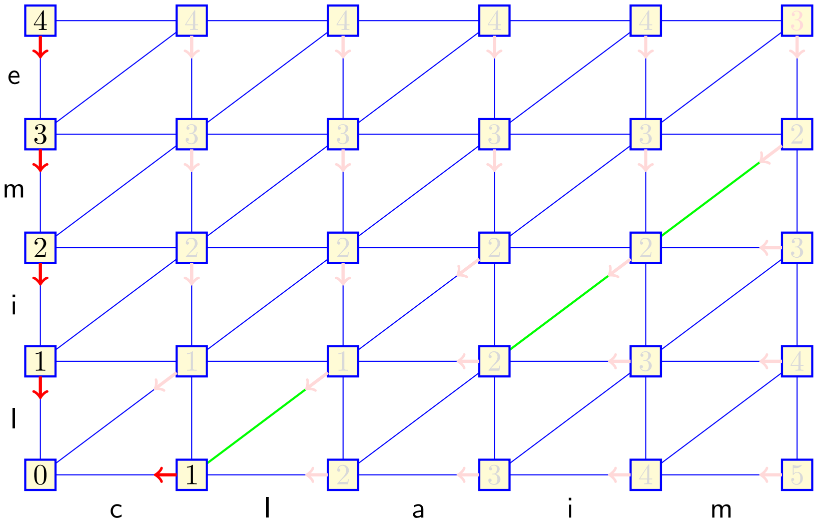
Levenshtein, or Edit Distance: DP Computation



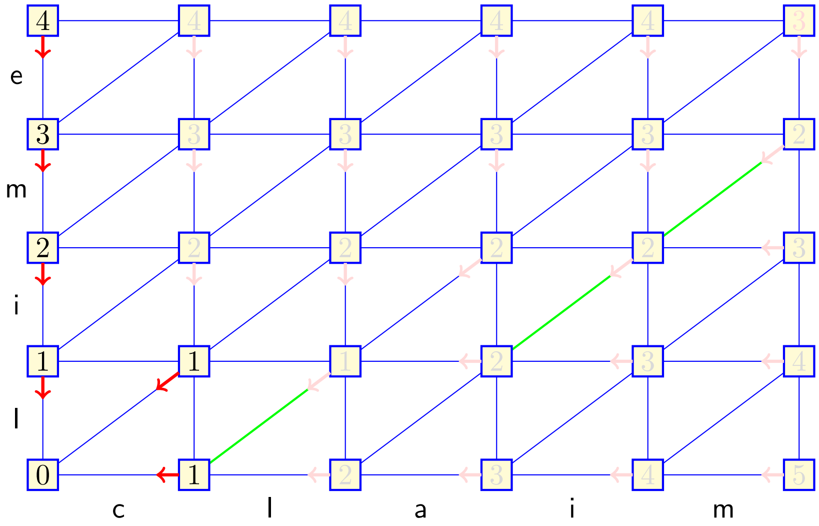
Levenshtein, or Edit Distance: DP Computation



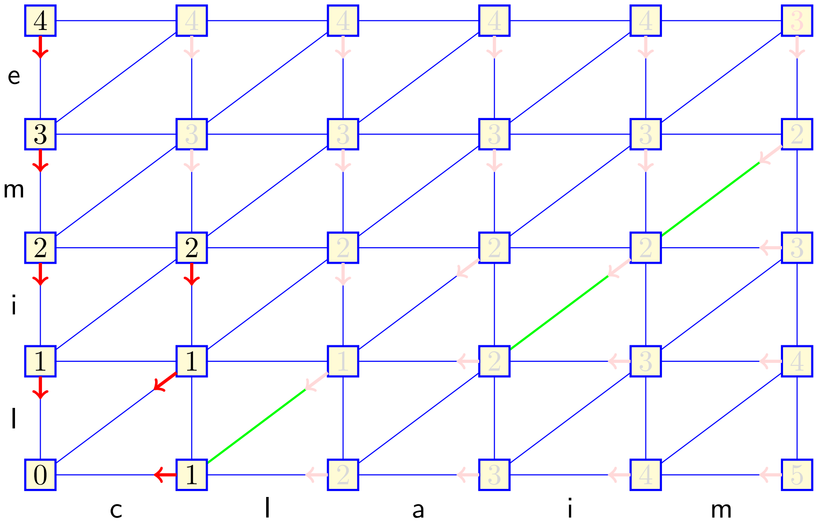
Levenshtein, or Edit Distance: DP Computation



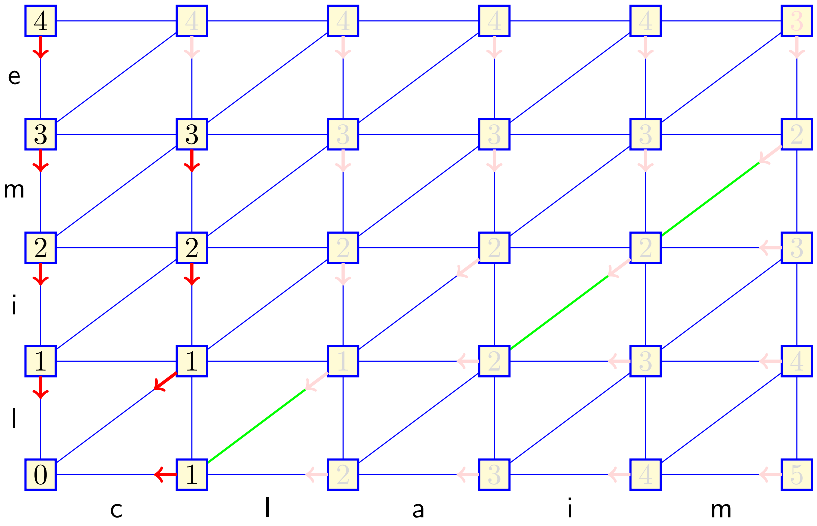
Levenshtein, or Edit Distance: DP Computation



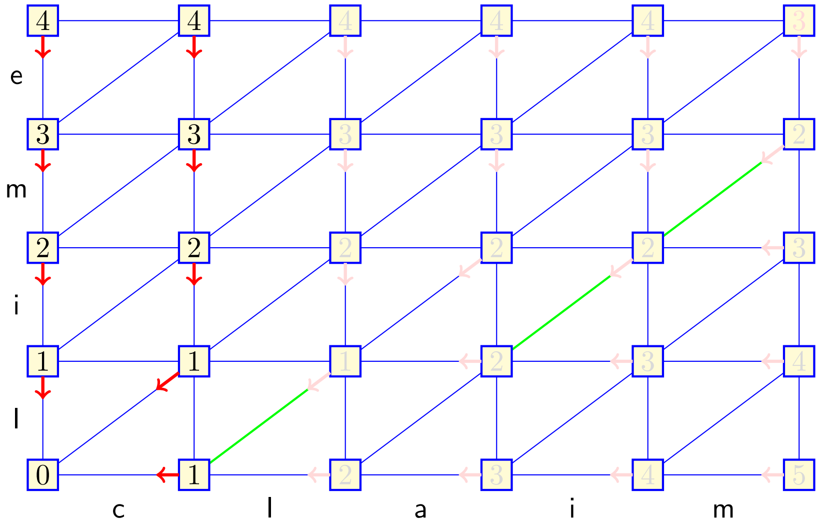
Levenshtein, or Edit Distance: DP Computation



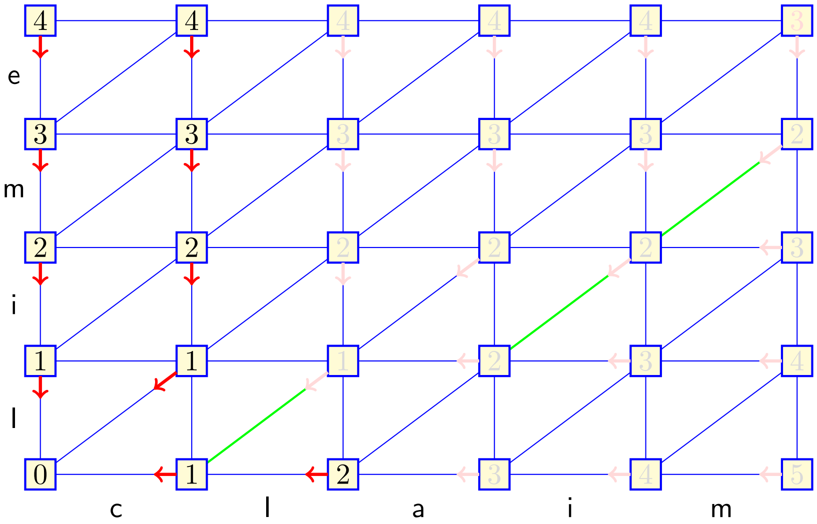
Levenshtein, or Edit Distance: DP Computation



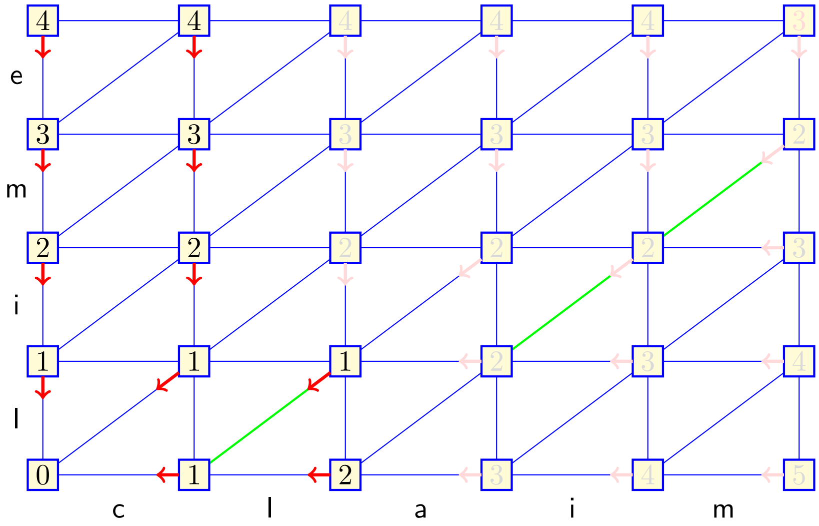
Levenshtein, or Edit Distance: DP Computation



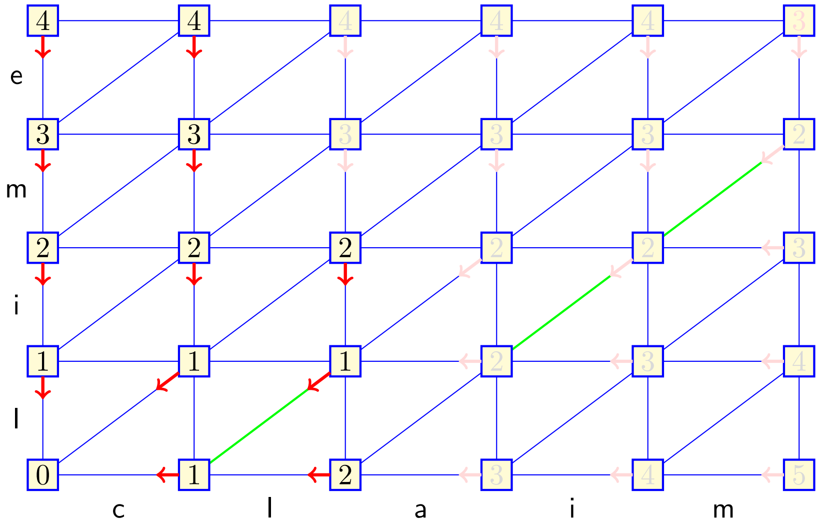
Levenshtein, or Edit Distance: DP Computation



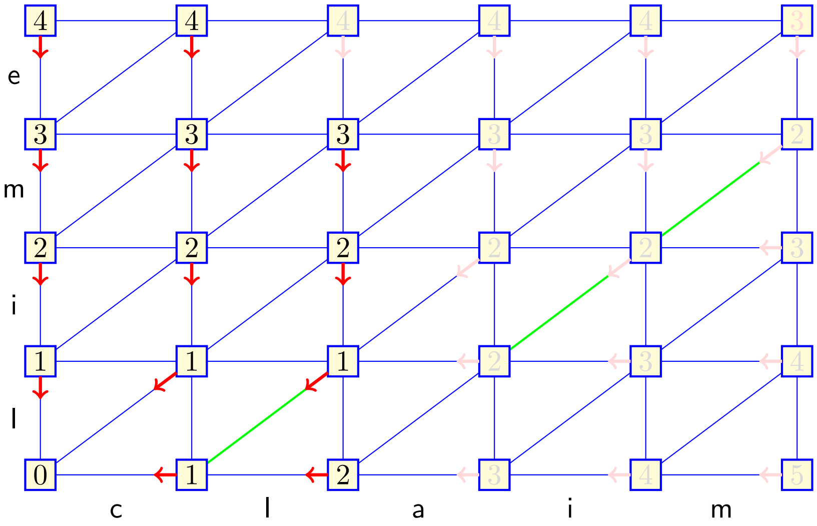
Levenshtein, or Edit Distance: DP Computation



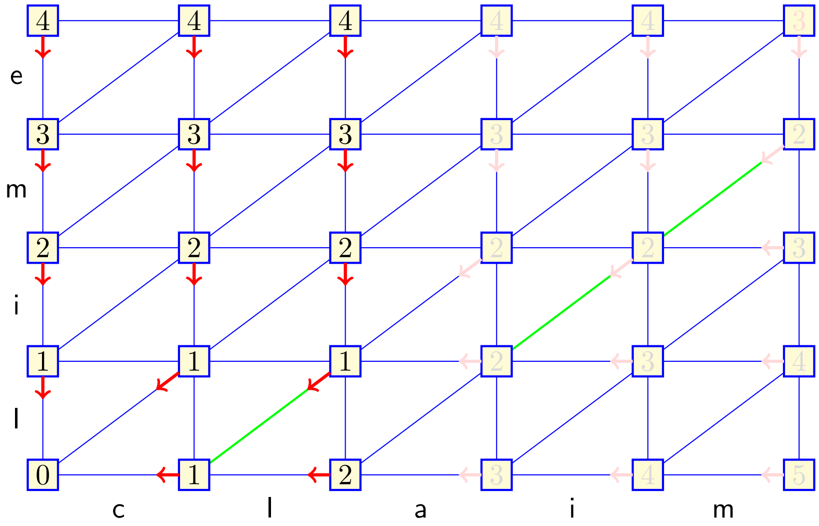
Levenshtein, or Edit Distance: DP Computation



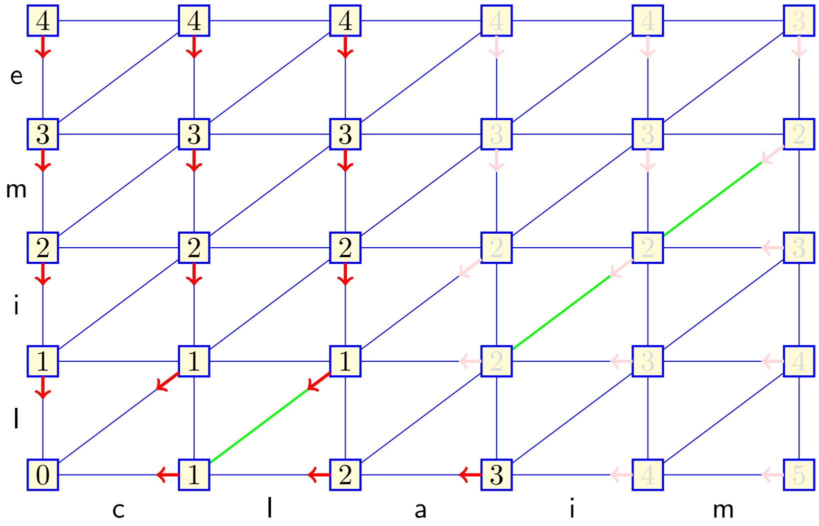
Levenshtein, or Edit Distance: DP Computation



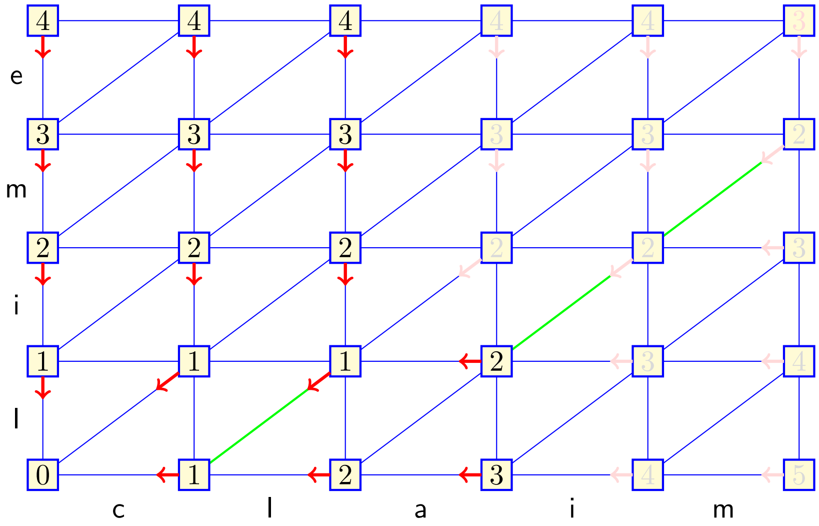
Levenshtein, or Edit Distance: DP Computation



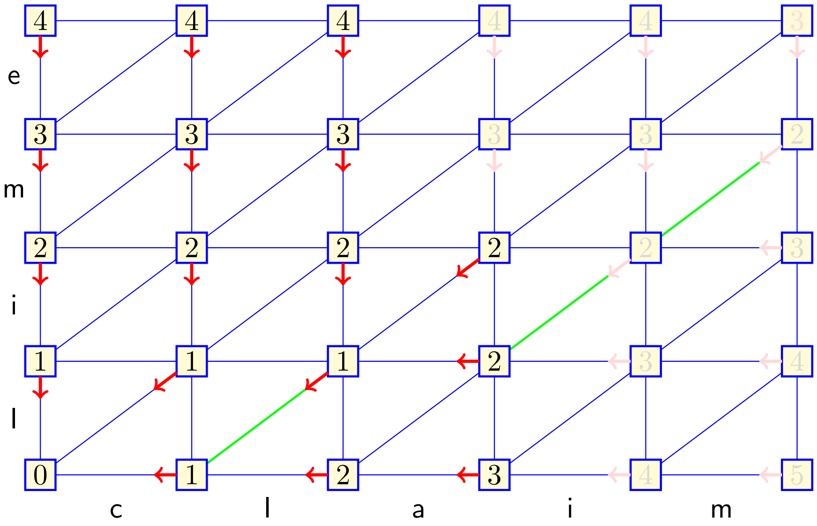
Levenshtein, or Edit Distance: DP Computation



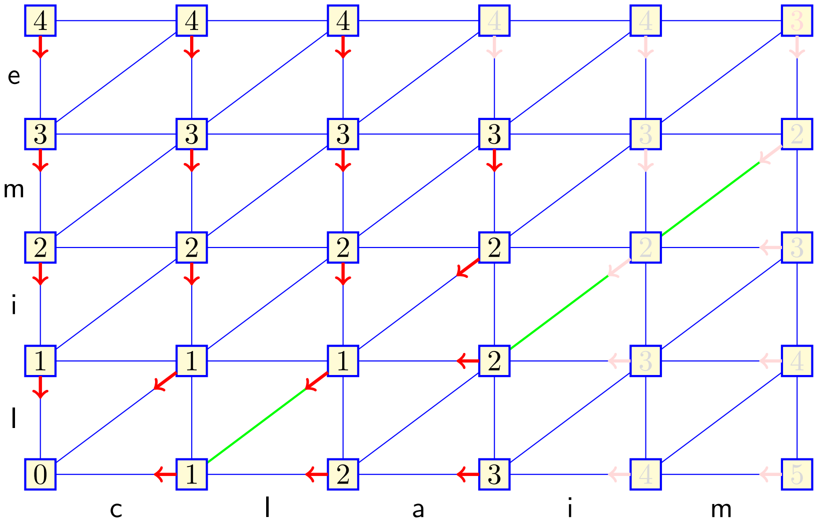
Levenshtein, or Edit Distance: DP Computation



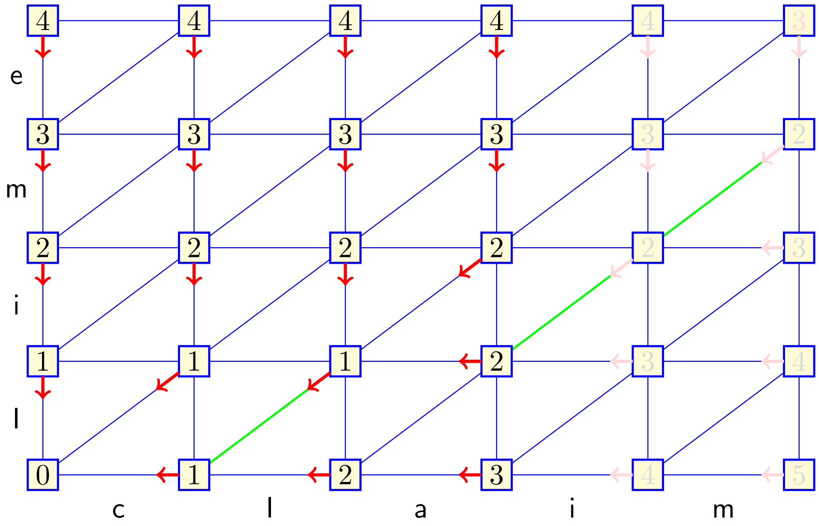
Levenshtein, or Edit Distance: DP Computation



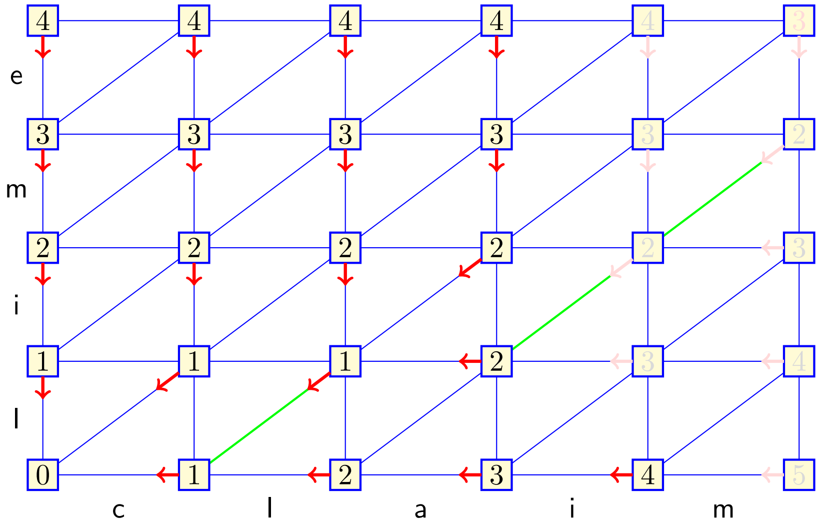
Levenshtein, or Edit Distance: DP Computation



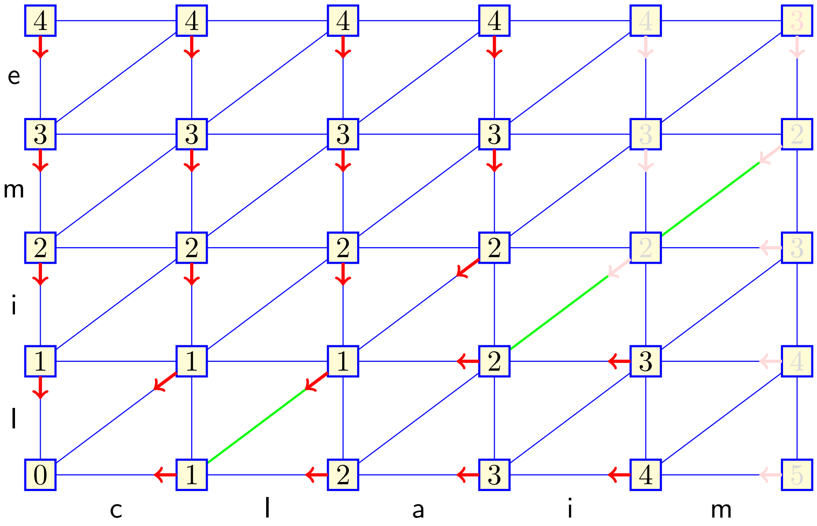
Levenshtein, or Edit Distance: DP Computation



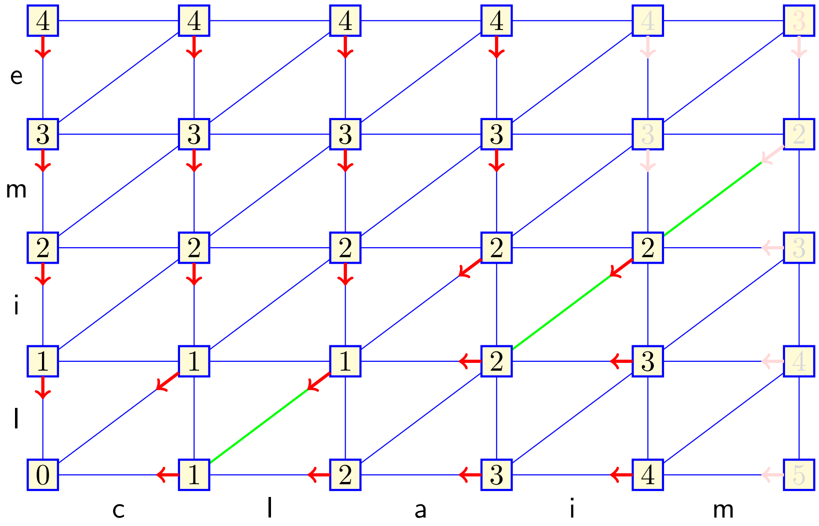
Levenshtein, or Edit Distance: DP Computation



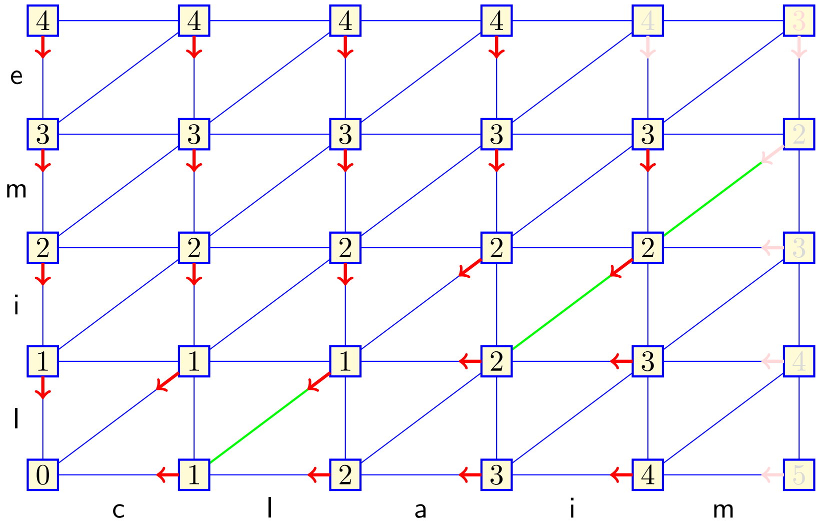
Levenshtein, or Edit Distance: DP Computation



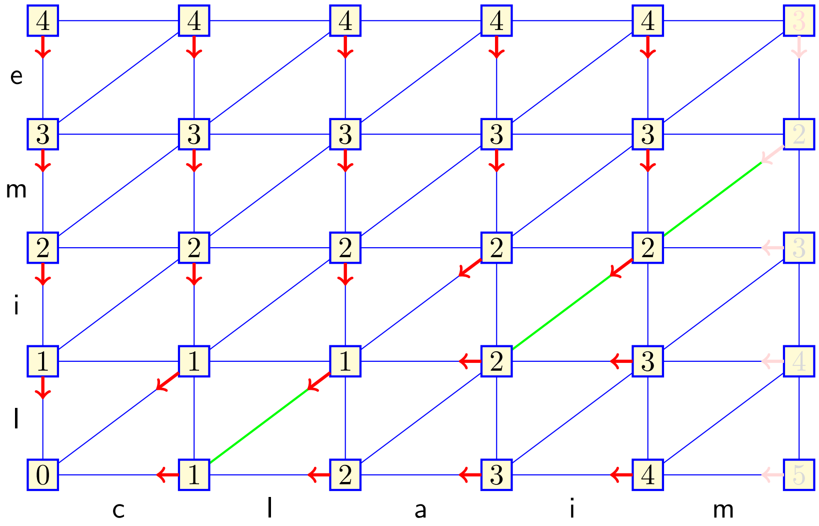
Levenshtein, or Edit Distance: DP Computation



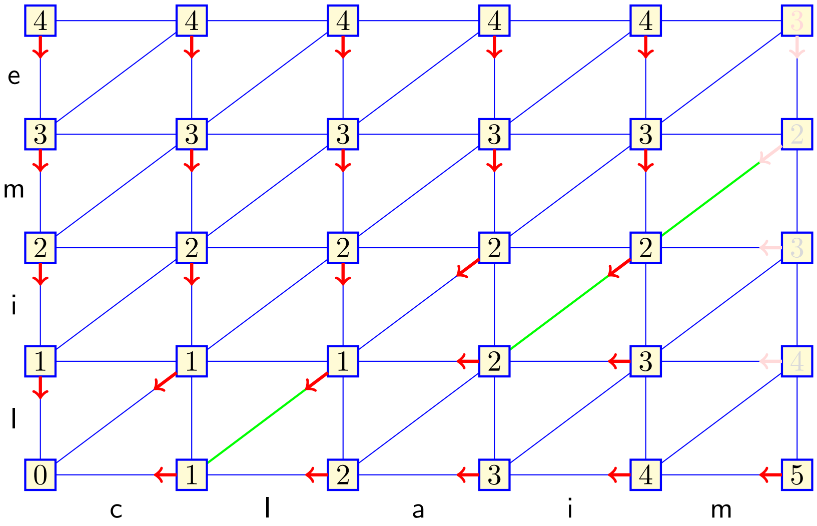
Levenshtein, or Edit Distance: DP Computation



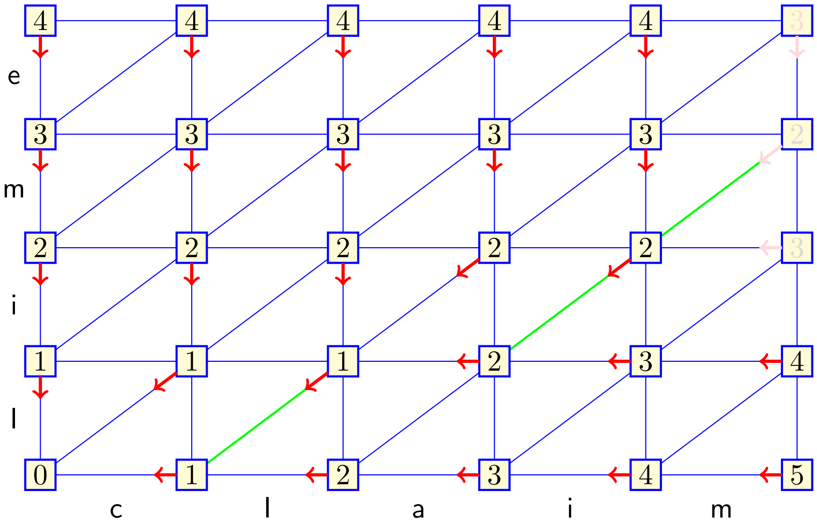
Levenshtein, or Edit Distance: DP Computation



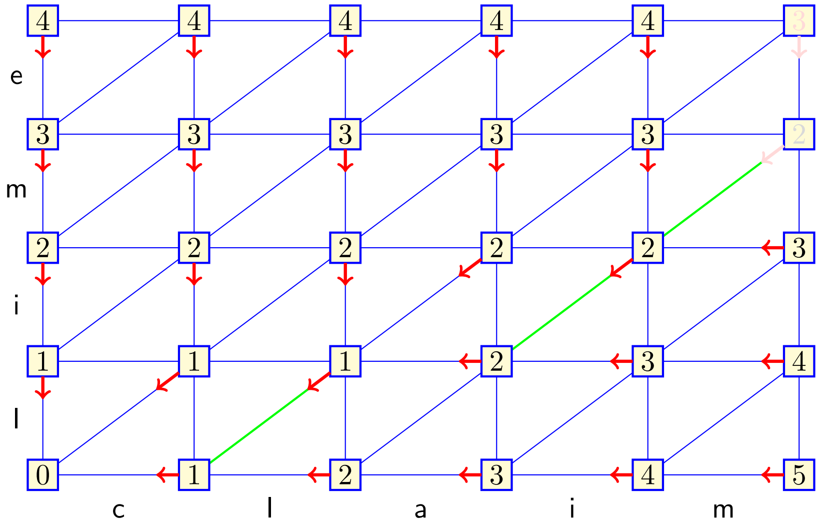
Levenshtein, or Edit Distance: DP Computation



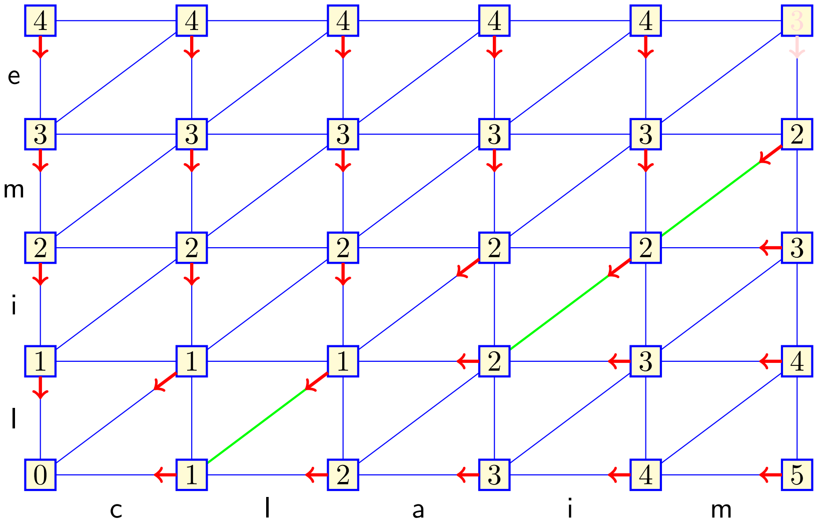
Levenshtein, or Edit Distance: DP Computation



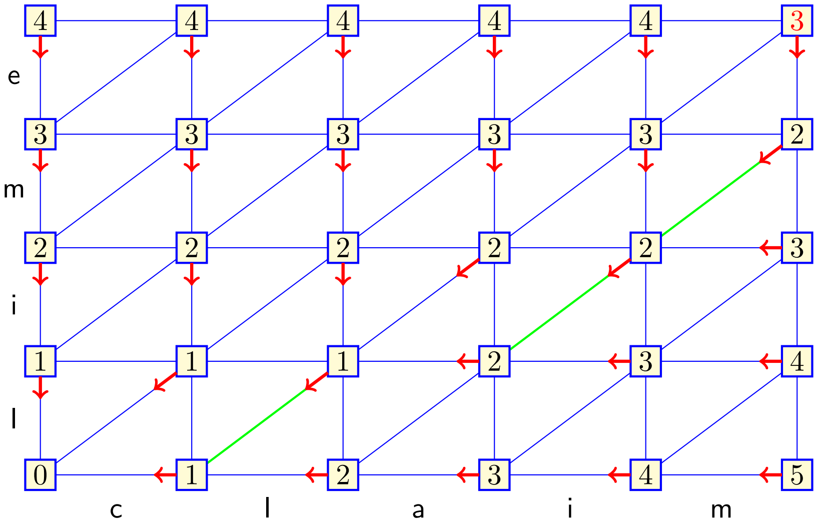
Levenshtein, or Edit Distance: DP Computation



Levenshtein, or Edit Distance: DP Computation



Levenshtein, or Edit Distance: DP Computation



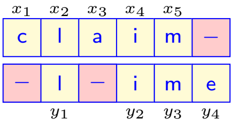
Sequence Alignment

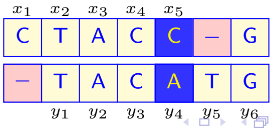
Given two strings $x = x_1 x_2 \ldots x_m$ and $y = y_1 y_2 \ldots y_n$, find their alignment of minimum cost

- Alignment $\mathbb{M}$: a set of ordered pairs $(x_i \mapsto y_j$, such that each item occurs in at most one pair and there are no crossings
- Pairs $x_i \mapsto y_j$ and $x_{i'} \mapsto y_{j'}$ cross if $i < i'$, but $j > j'$

$$\boxed{\operatorname{cost}(\mathbb{M}) = \underbrace{\sum_{(x_i, y_y) \in \mathbb{M}} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:x_i \text{ unmatched}} \delta + \sum_{j:y_j \text{ unmatched}} \delta}_{\text{gaps}}}$$

Example 1: $x = \text{claim}$; $y = \text{lime} \Rightarrow \mathbb{M} = \{x_2 \mapsto y_1, x_4 \mapsto y_2, x_5 \mapsto y_3\}$; $\text{cost} = 3\delta$
Example 2: $x = \text{CTACCG}$; $y = \text{TACATG} \Rightarrow$
$\mathbb{M} = \{x_2 \mapsto y_1, x_3 \mapsto y_2, x_4 \mapsto y3, x_5 \mapsto y_3, x_5 \mapsto y_4, x_6 \mapsto y_6\}$; $\text{cost} = \alpha_{\text{CA}} + 2\delta$




x_1 x_2 x_3 x_4 x_5
C T A C C – G
– T A C A T G
y_1 y_2 y_3 y_4 y_5 y_6

Sequence Alignment: Algorithm

```
Sequence-Alignment ( $x_1x_2\dots x_m$, $y_1y_2\dots y_n$, $\delta$, $\alpha$) {
  for i = 0 to m
     D[i, 0] = i$\delta$
  for j = 0 to n
     D[0, j] = j$\delta$
  for i = 1 to m
    for j = 1 to n
       D[i, j] = min( $\alpha[x_i, y_j]$ + D[i − 1, j − 1],
                      $\delta$ + D[i − 1, j], $\delta$ + D[i, j − 1] )
}
```

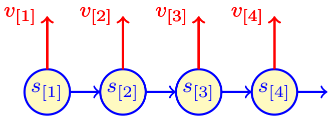
Time and space complexity $\Theta(mn)$

- English words: $m, n \approx 10$
- Computational biology: $m = n \approx 100,000$
  (10 billion operations is fine, but 10GB array?)

Viterbi Algorithm: Probabilistic Model

DP search for the most likely sequence of unobserved (hidden) states from a sequence of observations (signals)

- Each signal depends on exactly one corresponding hidden state
- The hidden states are produced by a first-order Markov model:
  - Set of the hidden states $\mathbb{S} = \{s_1, \ldots, s_n\}$
  - Transitional probabilities $P_{\mathrm{s}}(s_i|s_j) : i, j \in \{1, \ldots, n\}$
- Given the states, the signals are statistically independent
  - Set of the signals $\mathbb{V} = \{v_1, \ldots, v_m\}$
  - Observational probabilities $P_{\mathrm{o}}(v_j|s_i) : v_j \in \mathbb{V}; s_i \in \mathbb{S}$



Log-likelihood of a sequence of states $\mathbf{s} = s_{[1]}s_{[2]}\ldots s_{[K]}$, given a sequence of signals $\mathbf{v} = v_{[1]}v_{[2]}\ldots v_{[K]}$:

$$L(\mathbf{s}|\mathbf{v}) \equiv \log \Pr(\mathbf{s}|\mathbf{v}) \Rightarrow \Pr(\mathbf{s}|\mathbf{v}) \sim \Pr(\mathbf{s}, \mathbf{v}) = \Pr_{\mathrm{s}}(\mathbf{s}) \Pr_{\mathrm{o}}(\mathbf{v}|\mathbf{s})$$

http://en.wikipedia.org/wiki/Viterbi_algorithm

Maximum (Log-)Likelihood $\mathbf{s}^* = \arg\max_{\mathbf{s}\in\mathbb{S}^K} L(\mathbf{s}|\mathbf{v})$

1. $\mathbf{s} = s_{[1]}s_{[2]}\ldots s_{[K]}$ – a hidden (unobserved) Markov chain of states at steps $k = 1,\ldots,K$ with joint probability
$$\Pr_{\mathrm{s}}(\mathbf{s}) = \pi\left(s_{[1]}\right)\prod_{k=2}^{K} P_{\mathrm{s}}\left(s_{[k]}|s_{[k-1]}\right)$$
   - $\pi\left(s\right)$ – prior probability of state $s\in\mathbb{S}$ at step $k=1$
   - $P_{\mathrm{s}}\left(s|s'\right)$ – probability of transition from state s' to the next one, s
2. $\mathbf{v} = v_{[1]}v_{[2]}\ldots v_{[K]}$ – an observed sequence of conditionally independent signals with probability $\Pr(\mathbf{v}|\mathbf{s}) = \prod_{k=1}^{K} P_{\mathrm{o}}\left(v_{[k]}|s_{[k]}\right)$
   - $P_{\mathrm{o}}\left(v|s\right)$ – probability of observing $v\in\mathbb{V}$ in state $s\in\mathbb{S}$ at step k

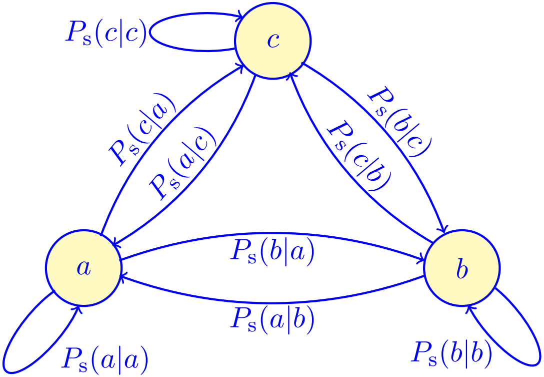
$$\mathbf{s}^* = \arg\max_{\mathbf{s}\in\mathbb{S}^K}\sum_{k=1}^{K}\left(\psi_k\left(s_{[k]}\right) + \varphi\left(s_{[k]}|s_{[k-1]}\right)\right)$$

$$\psi_k\left(s\right) = \begin{cases} \log\pi\left(s\right) + \log P_{\mathrm{o}}\left(v_{[k]}|s\right) & k=1;\; s\in\mathbb{S} \\ \log P_{\mathrm{o}}\left(v_{[k]}|s\right) & k>1;\; s\in\mathbb{S} \end{cases}$$

$$\varphi\left(s|s'\right) = \begin{cases} 0 & k=1;\; s\in\mathbb{S} \\ \log P_{\mathrm{s}}\left(s|s'\right) & k>1;\; s\in\mathbb{S} \end{cases}$$

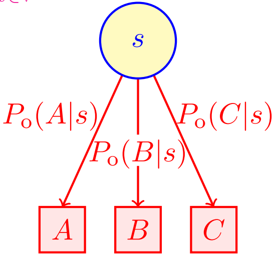
Probabilistic State Transitions and Signals for States

Example for $\mathbb{S} = \{a, b, c\}$, $\mathbb{V} = \{A, B, C\}$

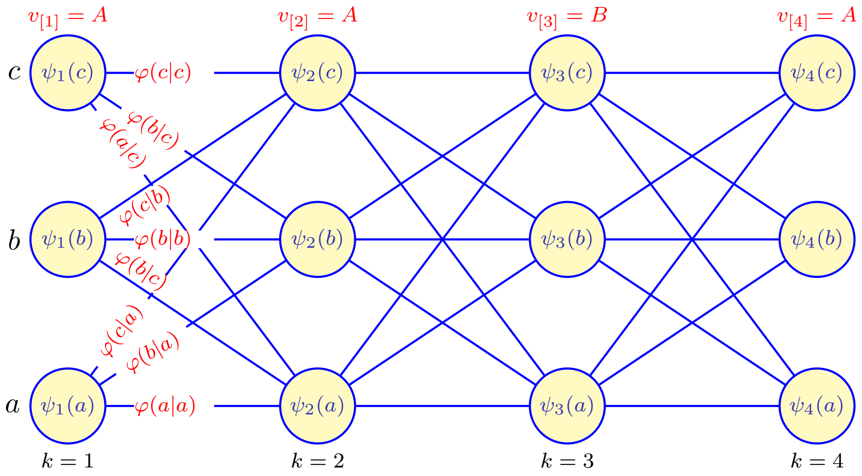


Non-deterministic (probabilistic) finite automaton (NFA) for state transitions at each step k; $\sum\limits_{s\in\mathbb{S}} P_\mathrm{s}(s|s') = 1$ for all $s' \in \mathbb{S}$

Probabilistic signal generator at each step k; $\sum\limits_{v\in\mathbb{V}} P_\mathrm{o}(v|s) = 1$ for all $s \in \mathbb{S}$



Graphical Model for $\mathbb{S} = \{a, b, c\}$ and Given Signals $\mathbf{v}$



Maximum (Log-)Likelihood via Dynamic Programming

Viterbi DP algorithm:

1. Initialisation: $k = 1$; $\Phi_1(s_{[1]}) = \psi_1(s_{[1]})$ for all $s_{[1]} \in \mathbb{S}$
2. Forward pass for $k = 2, \ldots, K$ and all $s_{[k]} \in \mathbb{S}$:

$$\Phi_k\left(s_{[k]}\right) = \psi_k(s_{[k]}) + \max_{s_{[k-1]}\in\mathbb{S}} \left\{\varphi(s_{[k]}|s_{[k-1]}) + \Phi_{k-1}\left(s_{[k-1]}\right)\right\}$$

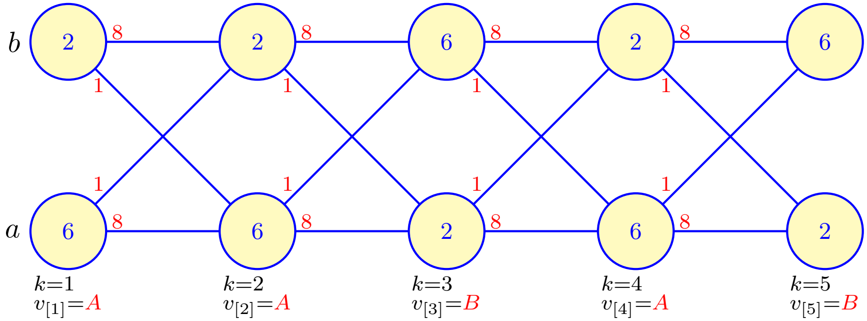
$$B_k\left(s_{[k]}\right) = \arg\max_{s_{[k-1]}\in\mathbb{S}} \left\{\varphi(s_{[k]}|s_{[k-1]}) + \Phi_{k-1}\left(s_{[k-1]}\right)\right\}$$

3. $k = K$: the maximum log-likelihood state
$$s^*_{[K]} = \arg\max_{s_{[k]}\in\mathbb{S}} \Phi_K(s_{[K]})$$
4. Backward pass for $k = K-1, \ldots, 1$: $s^*_{[k]} = B_{k+1}\left(s^*_{[k+1]}\right)$

Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases} ; \quad \text{for} \quad s, s' \in \mathbb{S}$$

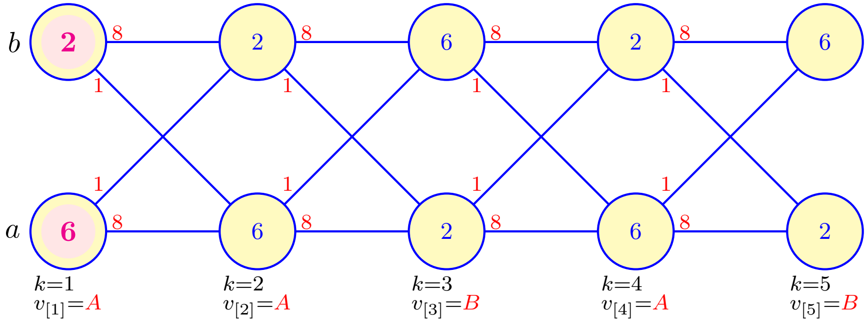
$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases} ; \quad \text{for} \quad s \in \mathbb{S}$$



Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases}; \quad \text{for} \quad s, s' \in \mathbb{S}$$

$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases}; \quad \text{for} \quad s \in \mathbb{S}$$

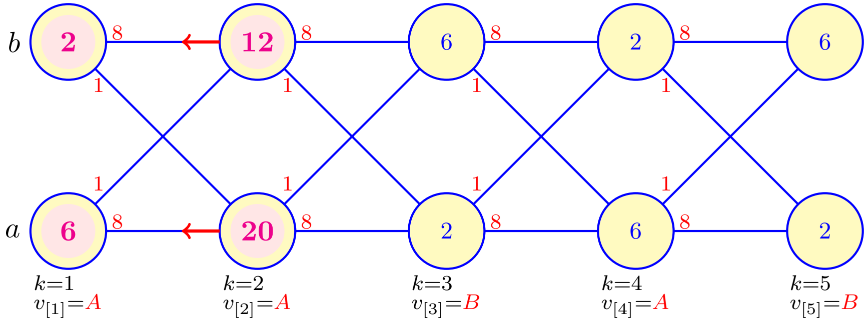


Step $k = 1$: Initialisation

Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases}; \quad \text{for} \quad s, s' \in \mathbb{S}$$

$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases}; \quad \text{for} \quad s \in \mathbb{S}$$

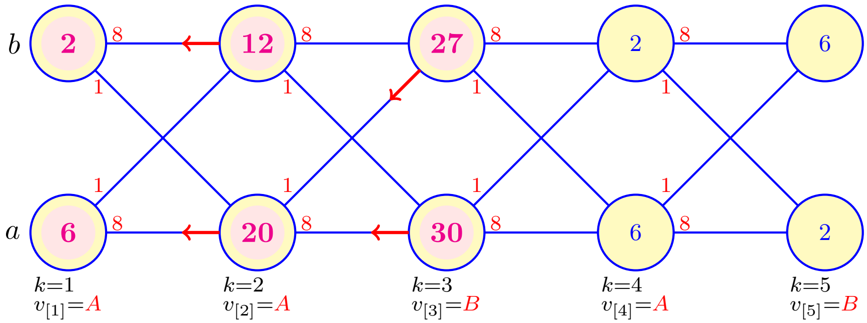


Step $k = 2$

Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases}; \quad \text{for} \quad s, s' \in \mathbb{S}$$

$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases}; \quad \text{for} \quad s \in \mathbb{S}$$

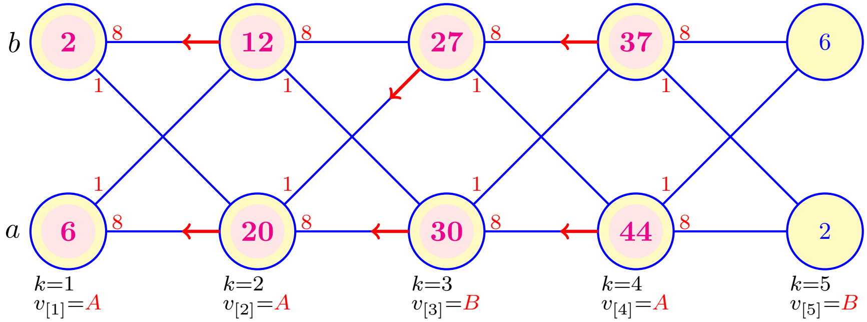


Step $k = 3$

Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases} ; \quad \text{for} \quad s, s' \in \mathbb{S}$$

$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases} ; \quad \text{for} \quad s \in \mathbb{S}$$

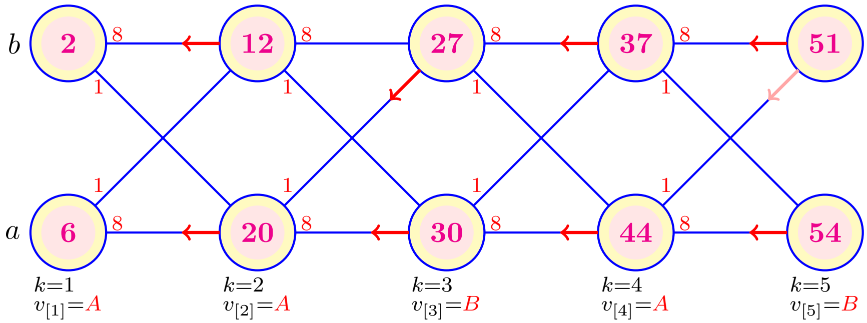


Step $k = 4$

Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases}; \quad \text{for} \quad s, s' \in \mathbb{S}$$

$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases}; \quad \text{for} \quad s \in \mathbb{S}$$

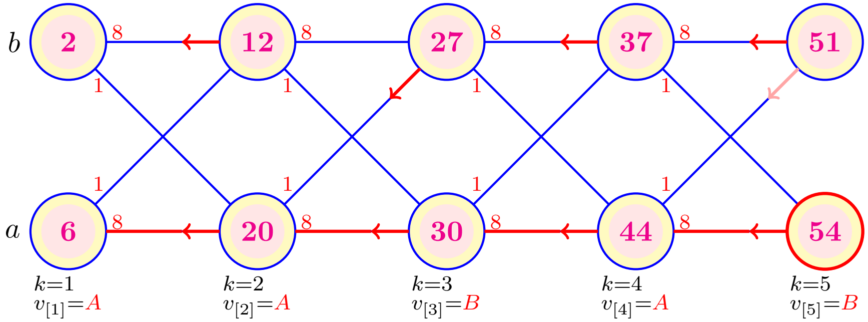


Step $k = 5$

Example: $\mathbb{S} = \{a, b\}$; $\mathbb{V} = \{A, B\}$; $\mathbf{v} = AABAB$

$$\varphi(s|s') = \begin{cases} 8 & s = s' \\ 1 & s \neq s' \end{cases} ; \quad \text{for} \quad s, s' \in \mathbb{S}$$

$$\psi_k(s) = \begin{cases} 6 & v|s \in \{A|a, B|b, C|c\} \\ 2 & \text{otherwise} \end{cases} ; \quad \text{for} \quad s \in \mathbb{S}$$



Backtracking: $\mathbf{s}^* = aaaaa$

“0–1” Knapsack Problem

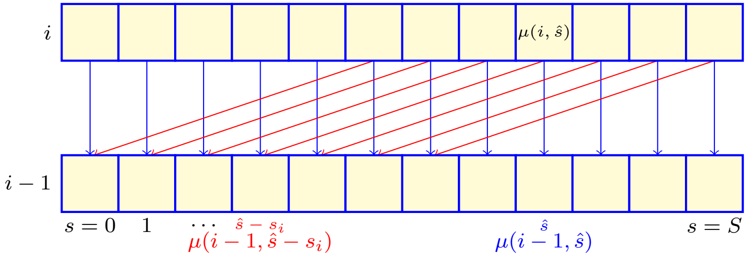
Maximise $\sum_{i=1}^{n} x_i v_i$ subject to $\sum_{i=1}^{n} x_i s_i \leq S$; $x_i \in \{0, 1\}$; $s_i, v_i, S > 0$

- DP solution of *pseudo-polynomial* time complexity, $O(nS)$
  - No contradiction to the NP-completeness of the problem: S is not polynomial in the length n of the problem's input
  - The length of S is proportional to the number of bits, i.e. $\log S$
  - Space complexity is $O(nS)$
    (or $O(S)$ if rewriting from $\mu(S)$ to $\mu(1)$ for each i)
- $\mu(i, s)$ – the maximum value that can be obtained by placing up to i items to the knapsack of size less than or equal to s
- DP solution uses a table $\mu(i, s)$ or $\mu(s)$ to store previous computations

"0–1" Knapsack Problem: DP Solution

Recursive definition of $\mu(i,s)$: $\mu(0,s)=\mu(i,0)=0$;

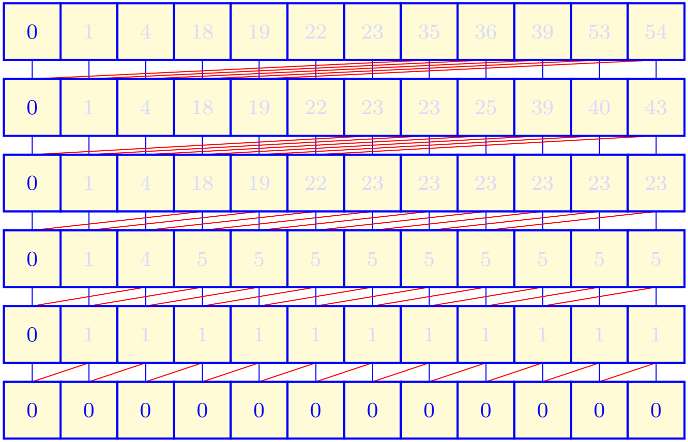
$$\mu(i,s)=\begin{cases}\mu(i-1,s) & \text{if} \quad s_i > s\\ \max\{\mu(i-1,s),\mu(i-1,s-s_i)+v_i\} & \text{if} \quad s_i\le s\end{cases}$$



“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7

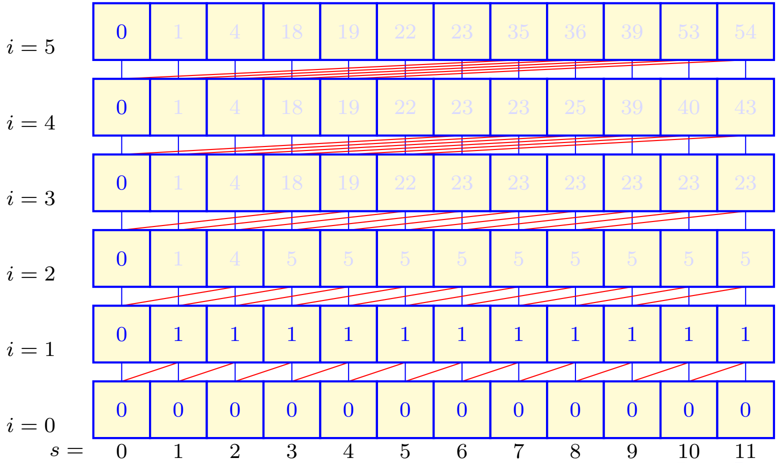


$i = 5$	0	1	4	18	19	22	23	35	36	39	53	54
$i = 4$	0	1	4	18	19	22	23	23	25	39	40	43
$i = 3$	0	1	4	18	19	22	23	23	23	23	23	23
$i = 2$	0	1	4	5	5	5	5	5	5	5	5	5
$i = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0
$s =$	0	1	2	3	4	5	6	7	8	9	10	11

“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7

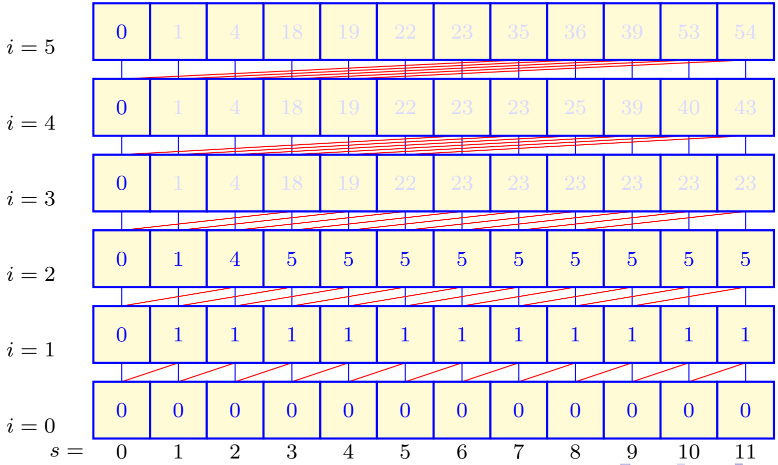


	0	1	2	3	4	5	6	7	8	9	10	11
$i = 5$	0	1	4	18	19	22	23	35	36	39	53	54
$i = 4$	0	1	4	18	19	22	23	23	25	39	40	43
$i = 3$	0	1	4	18	19	22	23	23	23	23	23	23
$i = 2$	0	1	4	5	5	5	5	5	5	5	5	5
$i = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0
$s =$	0	1	2	3	4	5	6	7	8	9	10	11

“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7

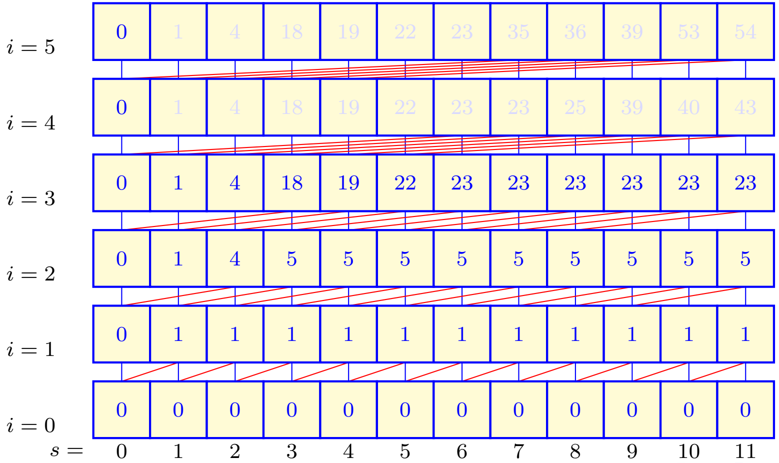


	0	1	2	3	4	5	6	7	8	9	10	11
$i = 5$	0	1	4	18	19	22	23	35	36	39	53	54
$i = 4$	0	1	4	18	19	22	23	23	25	39	40	43
$i = 3$	0	1	4	18	19	22	23	23	23	23	23	23
$i = 2$	0	1	4	5	5	5	5	5	5	5	5	5
$i = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0
$s =$	0	1	2	3	4	5	6	7	8	9	10	11

“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7

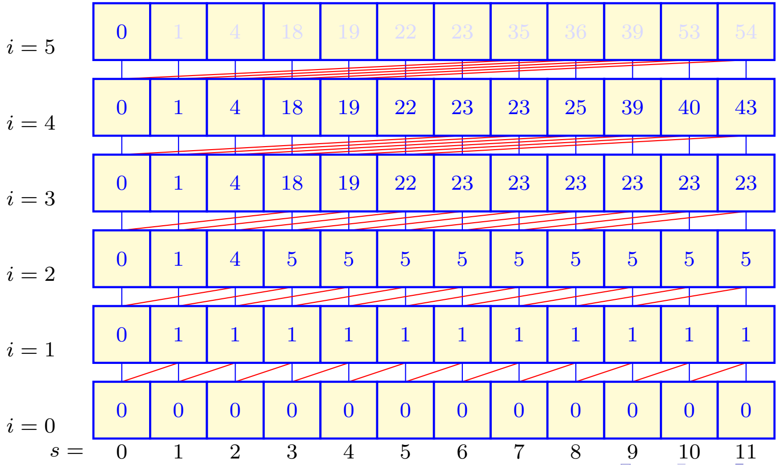


	0	1	2	3	4	5	6	7	8	9	10	11
$i = 5$	0	1	4	18	19	22	23	35	36	39	53	54
$i = 4$	0	1	4	18	19	22	23	23	25	39	40	43
$i = 3$	0	1	4	18	19	22	23	23	23	23	23	23
$i = 2$	0	1	4	5	5	5	5	5	5	5	5	5
$i = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0
$s =$	0	1	2	3	4	5	6	7	8	9	10	11

“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7

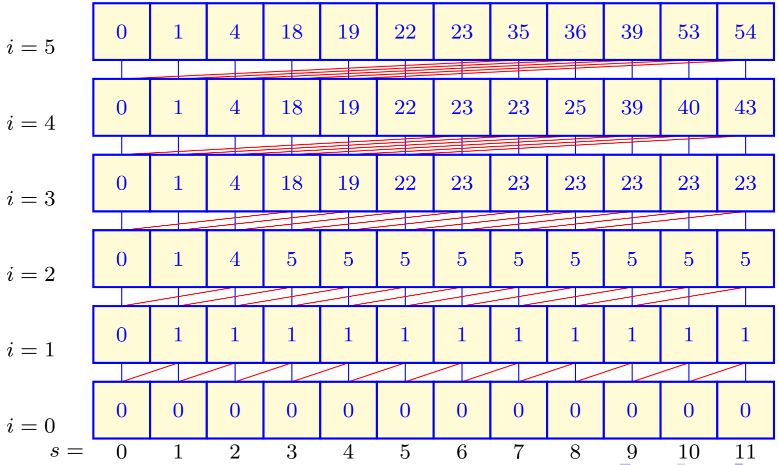


$i = 5$	0	1	4	18	19	22	23	35	36	39	53	54
$i = 4$	0	1	4	18	19	22	23	23	25	39	40	43
$i = 3$	0	1	4	18	19	22	23	23	23	23	23	23
$i = 2$	0	1	4	5	5	5	5	5	5	5	5	5
$i = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0
$s =$	0	1	2	3	4	5	6	7	8	9	10	11

“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7

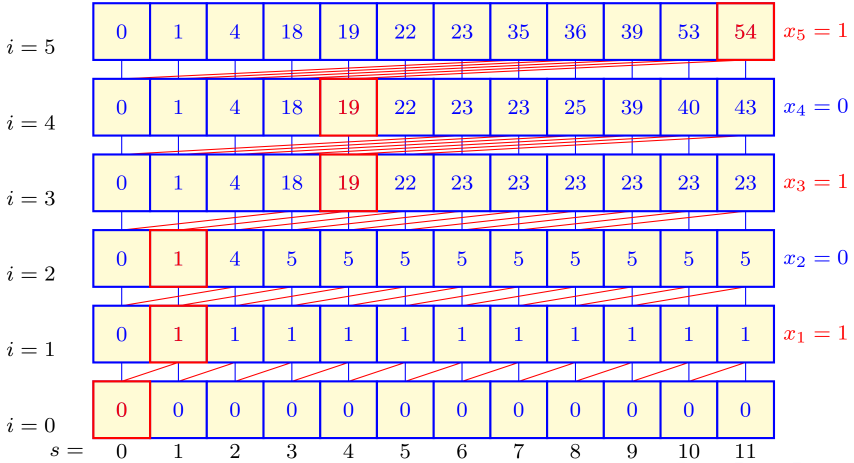


$i=5$	0	1	4	18	19	22	23	35	36	39	53	54
$i=4$	0	1	4	18	19	22	23	23	25	39	40	43
$i=3$	0	1	4	18	19	22	23	23	23	23	23	23
$i=2$	0	1	4	5	5	5	5	5	5	5	5	5
$i=1$	0	1	1	1	1	1	1	1	1	1	1	1
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0
$s=$	0	1	2	3	4	5	6	7	8	9	10	11

“0–1” Knapsack Problem: DP Solution

Example: n=5; S = 11;

i	1	2	3	4	5
v_i	1	4	18	21	35
s_i	1	2	3	6	7



	0	1	2	3	4	5	6	7	8	9	10	11	
$i = 5$	0	1	4	18	19	22	23	35	36	39	53	54	$x_5 = 1$
$i = 4$	0	1	4	18	19	22	23	23	25	39	40	43	$x_4 = 0$
$i = 3$	0	1	4	18	19	22	23	23	23	23	23	23	$x_3 = 1$
$i = 2$	0	1	4	5	5	5	5	5	5	5	5	5	$x_2 = 0$
$i = 1$	0	1	1	1	1	1	1	1	1	1	1	1	$x_1 = 1$
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0	
$s =$	0	1	2	3	4	5	6	7	8	9	10	11	