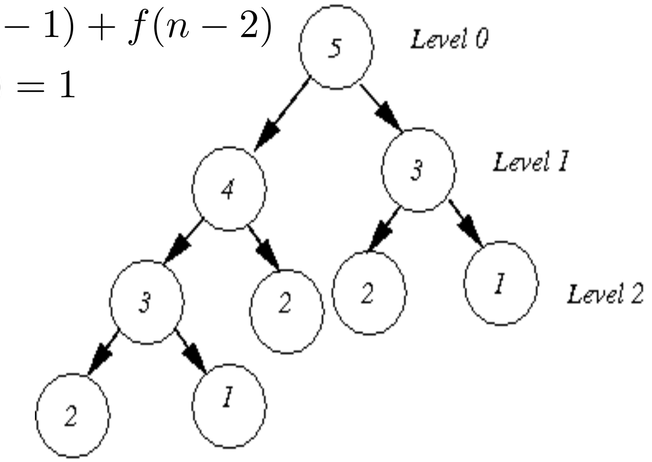
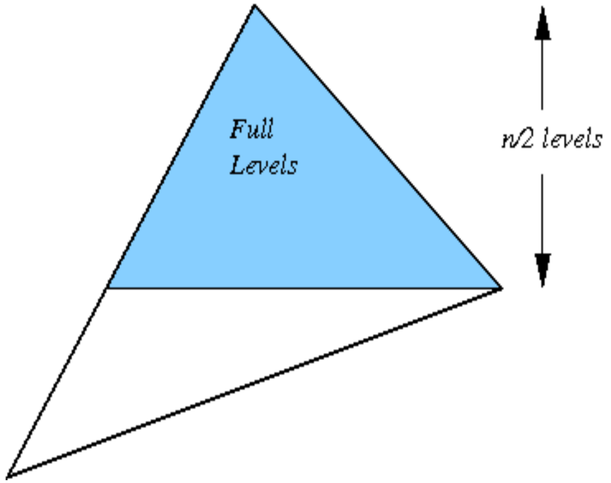
Dynamic Programming 101

- Divide-and-Conquer = single divide + disjoint conquer + single combine
- Dynamic Programming = multiple divides + memoized conquer + summarized combine
- the simplest example is Fibonacci

$$f(n) = f(n-1) + f(n-2)$$
$$f(1) = f(2) = 1$$






```
def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)
```

naive recursion without memoization: $O(1.618...^n)$

DP2: bottom-up: $O(n)$

```
def fib0(n):
  a, b = 1, 1
  for i in range(3, n+1):
    a, b = a+b, a
  return a
```

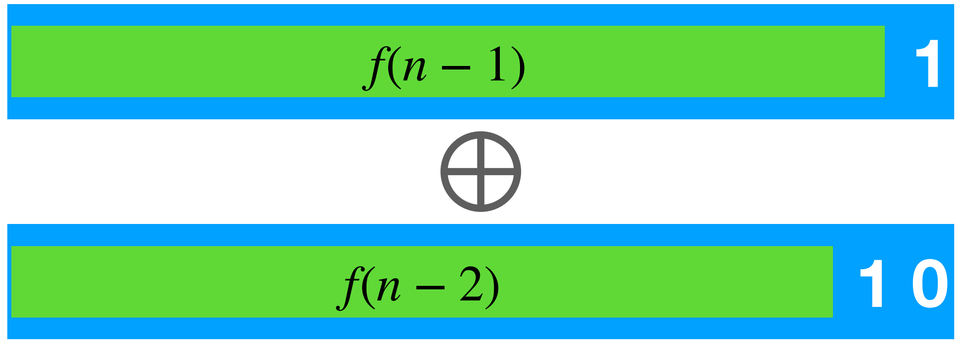
```
def fib0(n):
  f = [1, 1]
  for i in range(3, n+1):
    f.append(f[-1]+f[-2])
  return f[-1]
```

DP1: top-down with memoization: $O(n)$

```
fibs={1:1, 2:1} # hash table (dict)
def fib1(n):
  if n not in fibs:
    fibs[n] = fib1(n-1) + fib1(n-2)
  return fibs[n]
```

Number of Bitstrings

- number of *n*-bit strings that do not have 00 as a substring
  - e.g. *n*=1: 0, 1; *n*=2: 01, 10, 11; *n*=3: 010, 011, 101, 110, 111
  - what about *n*=0?
  - last bit "1" followed by *f*(*n*-1) substrings
  - last two bits "01" followed by *f*(*n*-2) substrings



$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 2, \quad f(0) = 1$$

Max Independent Set (MIS)

- max weighted independent set on a linear-chain graph
  - e.g. **9** — 10 — **8** — 5 — 2 — **4** ; best MIS: [9, 8, 4] = 21 (vs. greedy: [10, 5, 4] = 19)
  - subproblem: *f*(*i*) -- max independent set for a[1]..a[*i*] (1-based index)

$$f(i) = \max\{f(i-1),\ f(i-2) + a[i]\}$$

$$b(i) = [f(i) \neq f(i-1)]: \text{ take } a[i] \text{ for } f(i)?$$

$f(0) = 0; f(1) = a[1]$?
No! $f(1) = \max\{a[1], 0\}$
or even better: $f(0) = 0;\ f(-1) = 0$

i	-1	0	1	2	3	4	5	6
a[i]			9	10	8	5	2	4
f(i)	0	0	9	10	17	17	19	21
b(i)			T	T	T	F	T	T
back track	*		take *		take *	not *		take *

best value
backpointer
start here

recursively backtrack
the optimal solution

MIS

$$f(n) = \max \begin{cases} f(n-1) + 0 \\ f(n-2) + a[n] \end{cases}$$

bitstrings

$$f(n) = + \begin{cases} f(n-1) \times 1 \\ f(n-2) \times 1 \end{cases}$$

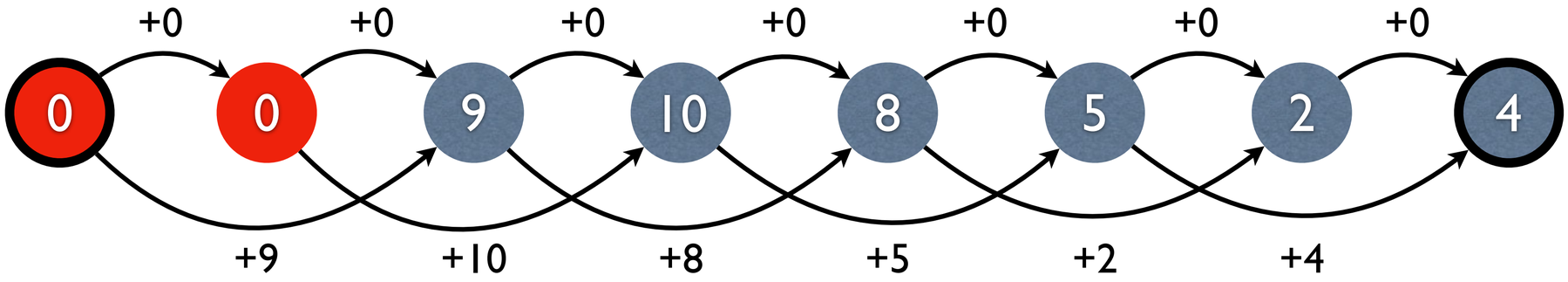
cost / reward

summary operator $\oplus$ (across divides)

combination operator $\otimes$ (within a divide)

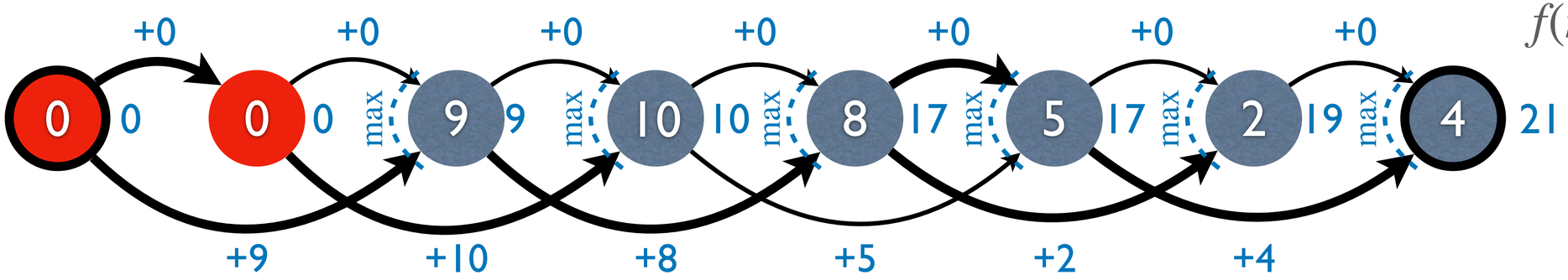
Graph Interpretation of DP

- MIS: longest path between source and target (see lecture video)
  - each node i has two incoming edges: $(i-2) \xrightarrow{a[i]} i$ (take) and $(i-1) \xrightarrow{0} i$ (not take)
  - $f(i)$: longest path between source and node i
- fibonacci & bitstrings: number of paths between source and target



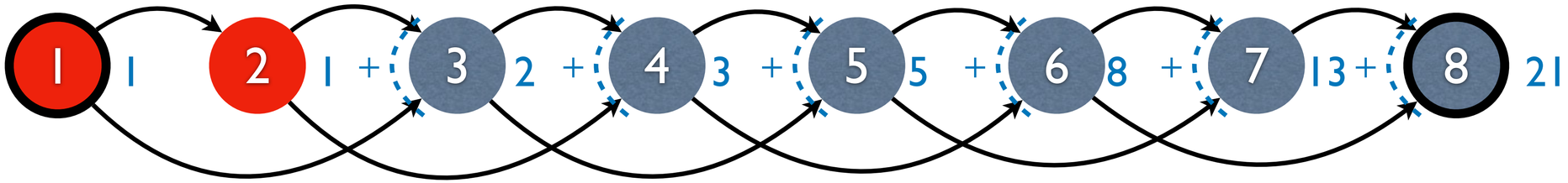
$$f(n) = \max \begin{cases} f(n-1) + 0 \\ f(n-2) + a[n] \end{cases}$$

$$f(n) = + \begin{cases} f(n-1) \times 1 \\ f(n-2) \times 1 \end{cases}$$



Graph Interpretation of DP

- MIS: longest path between source and target (see lecture video)
  - each node i has two incoming edges: $(i-2) \xrightarrow{a[i]} i$ (take) and $(i-1) \xrightarrow{0} i$ (not take)
  - $f(i)$: longest path between source and node i
- fibonacci & bitstrings: number of paths between source and target
  - unweighted graph (no cost on edge)



Summary

- Divide-and-Conquer = single divide + disjoint conquer + single combine
- Dynamic Programming = multiple divides + memoized conquer + summarized combine
- overlapping subproblems due to multiple divides (still disjoint subproblems within each divide)
- two implementation styles
  - 1. recursive top-down + memoization
  - 2. bottom-up
- backtracking to recover best solution for optimization problems
  - 1. backpointers (recommended); 2. store subsolutions (not recommended — often slows down); 3. recompute on-the-fly
- two operators: $\oplus$ for summary (across multiple divides) and $\otimes$ for combine (within a divide)
- counting problems vs. optimization problems ("cost-reward model")
- three steps in solving a DP problem
  - define the subproblem
  - recursive formula
  - base cases

cost reward

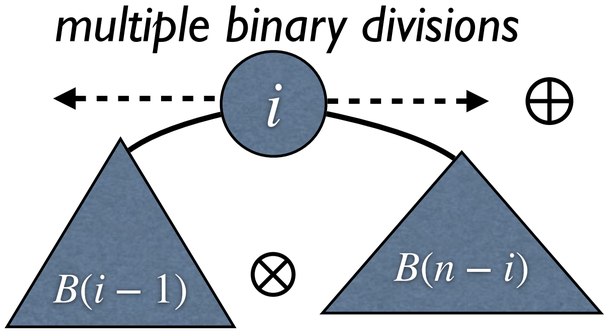
$$f(n) = \max \begin{cases} f(n-1) + 0 \\ f(n-2) + a[n] \end{cases}$$

summary operator $\oplus$ (across divides)

combination operator $\otimes$ (within a divide)

Deeper Understanding of DP

- divide-n-conquer
  - single division, independent conquer, combine
- DP = divide-n-conquer with multiple divisions
  - for each possible division
    - divide
    - conquer with memoization
    - combine subsolutions using the combination operator $\otimes$



  - summarize over all possible divisions using the summary operator $\oplus$

	$\oplus$	$\otimes$
Fib	+	x
MIS	max	+
# BSTs	+	x
knapsack	max	+
shortest path	min	+

- multiple divisions => overlapping subproblems
  - each single division => independent subproblems!

$$B(n) = \oplus_{i=1}^{n} \left(B(i-1) \otimes B(n-i) \right)$$

$$B(0) = 1$$

Unary vs. Binary Divisions

$(a) : T(n) = 2T(n/2) + \ldots$

$(b) : T(n) = T(n-1) + \ldots$

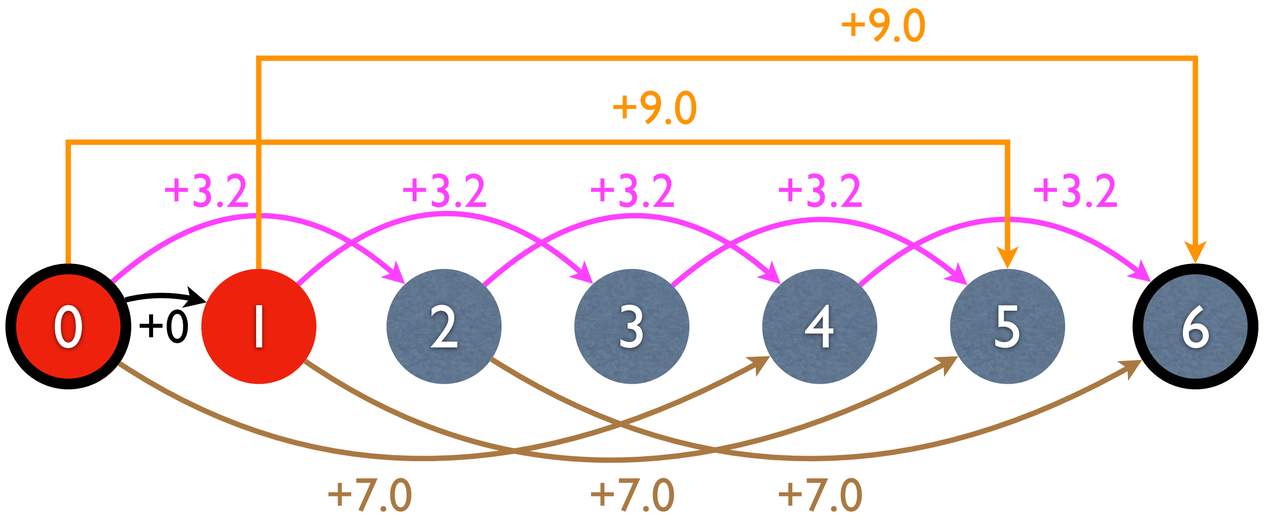
$(c) : T(n) = T(n/2) + \ldots$

	branching (binary division)	one-sided (unary division)
divide-n-conquer	quicksort, best-case	quicksort, worst-case (*b*)
	mergesort	quickselect: worst (*b*), best (*c*)
	(balanced) tree traversal (DFS)	binary search: (*c*)
	heapify (top-down)	search in BST: worst (*b*), best (*c*)
DP	# of BSTs (hw5), *midterm*	Fib, # of bitstrings (hw5)…
	optimal BST, *final*	max indep. set (hw5)
	RNA folding (hw10)	knapsack (hw6), *midterm*
	context-free parsing	Viterbi (hw8), *final*
	matrix-chain multiplication, …	LCS, LIS, edit-distance,…

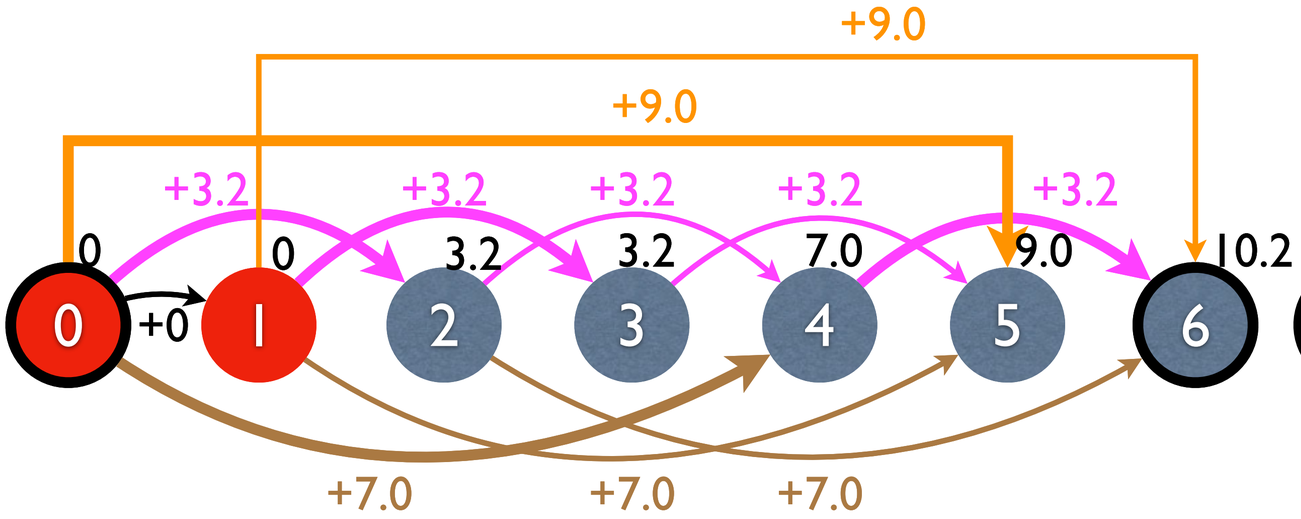
Two Divisions vs. Multiple Divisions (# of Choices)

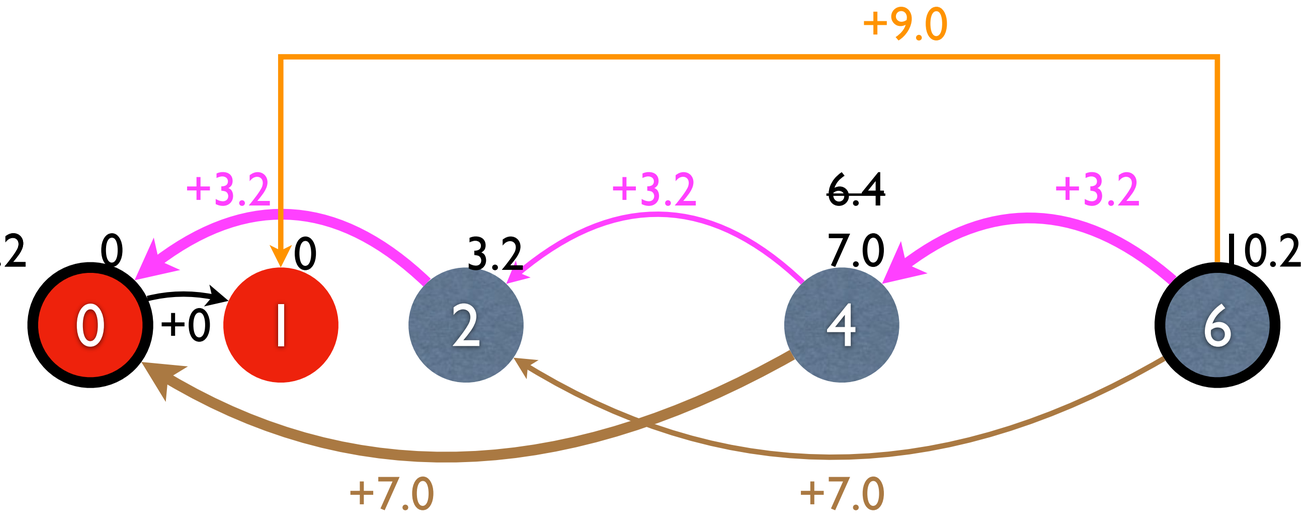
	two divisions	multiple divisions
DP	Fib, # of bitstrings (hw5)…	# of BSTs (hw5)
	max indep. set (hw5)	unbounded knapsack (hw6)
	0-1 knapsack (hw6)	bounded knapsack (hw6)
		Viterbi (hw8)
		RNA folding (hw10)

Graph Interpretation of Unbounded Knapsack



i	w_i	v_i
1	2	3.2
2	5	9.0
3	4	7.0





Knapsack: 0-1

- KT slides (DP chapter)

i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

weight limit w

subset of items 1, …, i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
{ }	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

OPT(i, w) = optimal value of knapsack problem with items 1, …, i, subject to weight limit w

37

Knapsack: 0-1

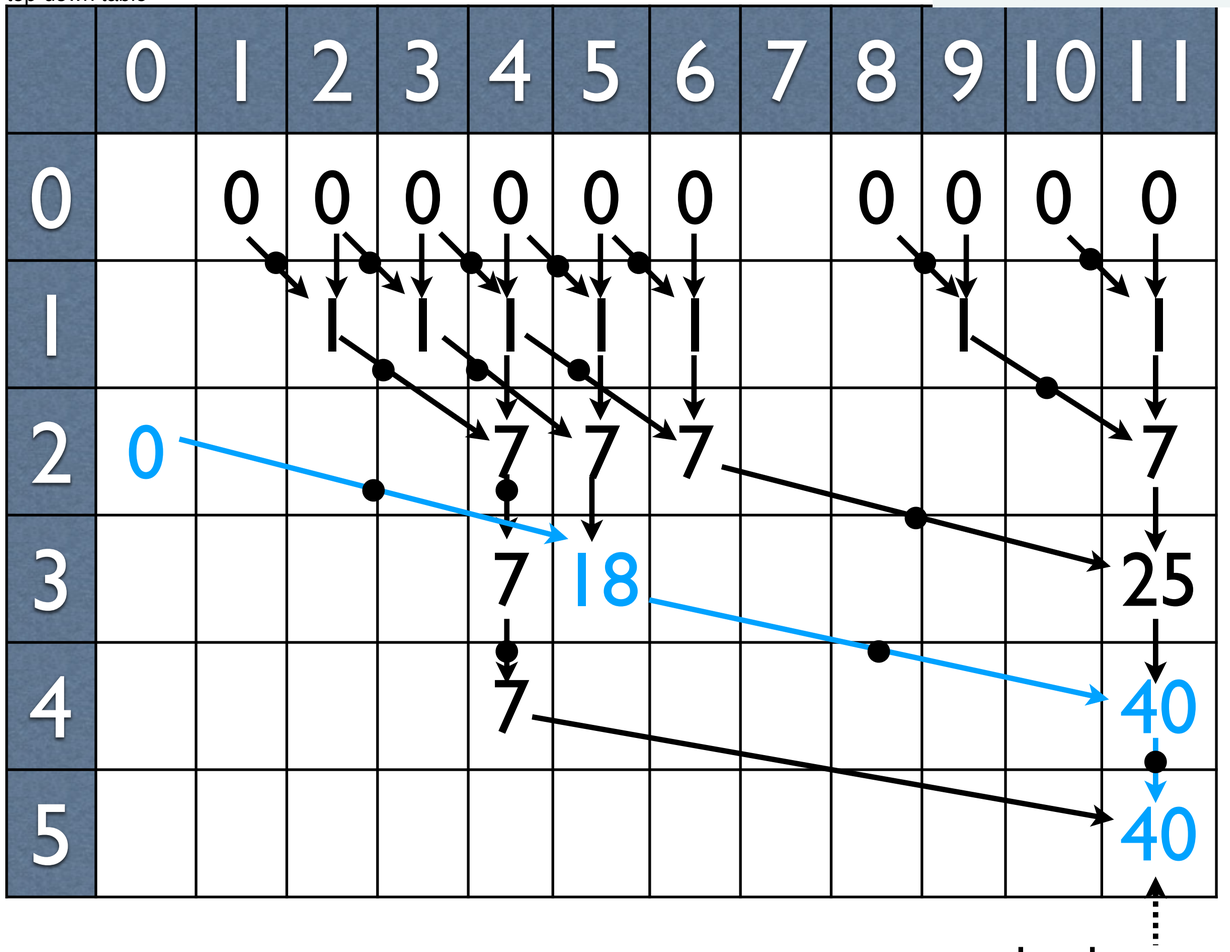
i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

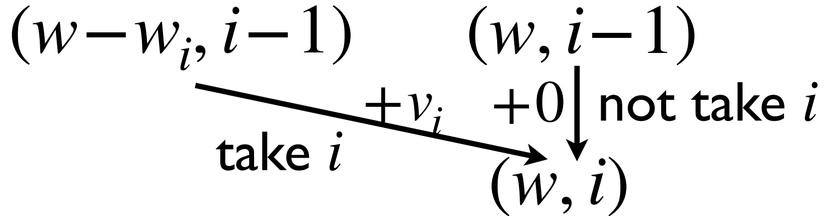
- bottom-up (dense) vs top-down (sparse)

bottom-up table

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

top-down table

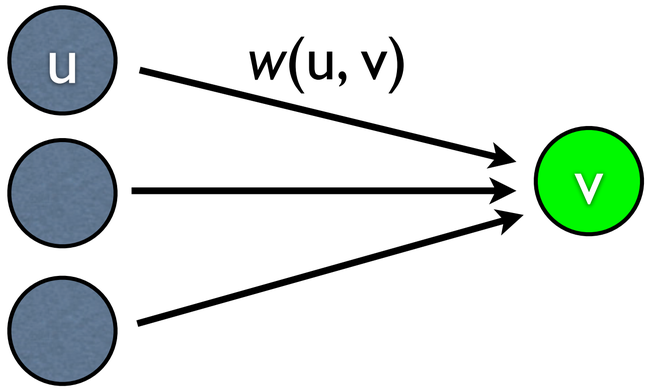




- backpointer → global best path

Viterbi Algorithm for DAGs

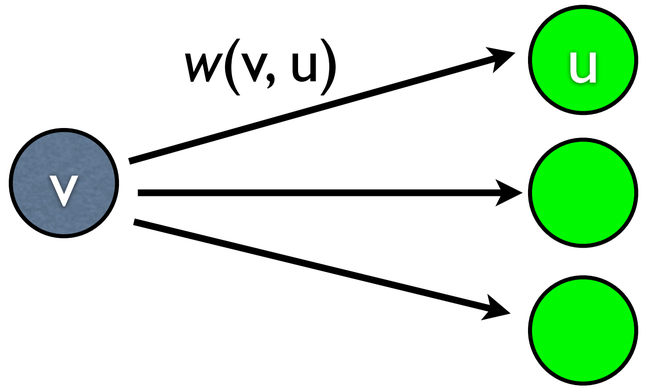
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
     - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
     - key observation: d(u) is fixed to optimal at this time



- time complexity: O(V + E)

Variant 1: forward-update

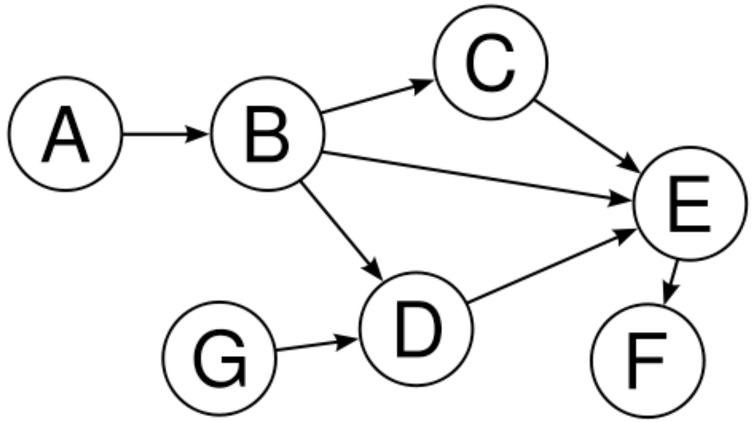
1. topological sort
2. visit each vertex v in sorted order and do updates
   - key observation: d(v) is fixed to optimal at this time
   - for each outgoing edge (v, u) in E
     - use d(v) to update d(u): $d(u) \oplus = d(v) \otimes w(v, u)$



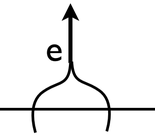
- time complexity: O(V + E)

Variant 2: Recursive Descent

- Top-down Recursion + Memoization = Bottom-up
- Start from the target vertex, going backwards
  - remember each visited vertex
- Sometimes easier to implement
- There is a tradeoff b/w top-down and bottom-up

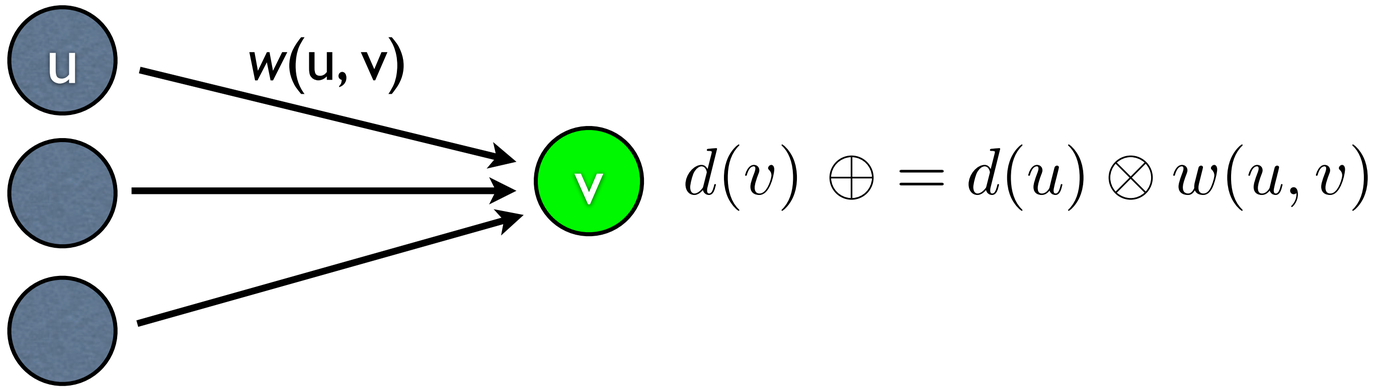


One-way vs. Two-way Divides (Graph vs. Hypergraph)

	two-way (binary divide)	one-way (unary divide)
divide-n-conquer	quicksort, best-case	quicksort, worst-case
	mergesort	quickselect
	tree traversal (DFS)	binary search
	heapify (top-down)	search in BST
DP	# of BSTs (hw5)	Fib, # of bitstrings (hw5)…
	 optimal BST	max indep. set (hw5)
	RNA folding (hw10)	knapsack (all kinds, hw6)
	context-free parsing	Viterbi (hw8)
	matrix-chain multiplication, …	LCS, LIS, edit-distance,…

Viterbi Algorithm for DAGs

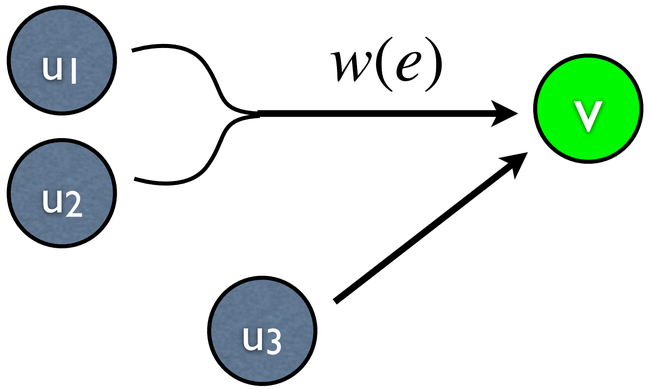
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



- time complexity: O(V + E)

Generalized Viterbi for DAHs (Hypergraphs)

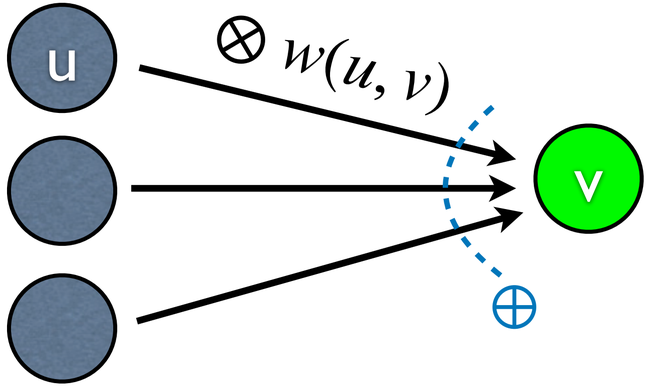
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming hyperedge e = ((u1, .., u|e|), v, w(e))
   - use d(ui)'s to update d(v)
   - key observation: d(ui)'s are fixed to optimal at this time



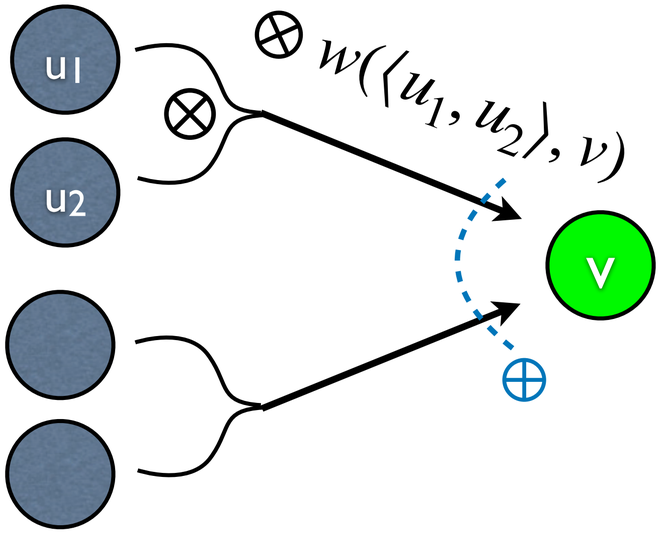
$$d(v) \oplus = d(u_1) \otimes d(u_2) \otimes w(e)$$

- time complexity: O(V + E) (assuming constant arity)

Graphs vs. Hypergraphs



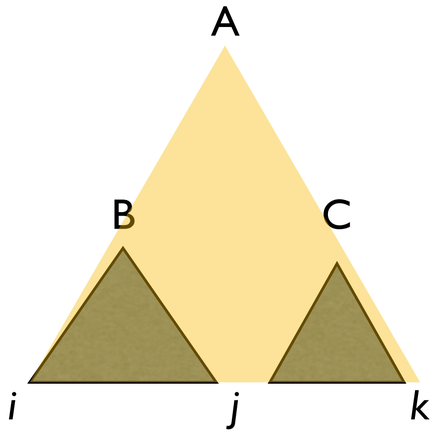
$$d(v) = \oplus_{(u,v)\in E} [d(u) \otimes w(u, v)]$$

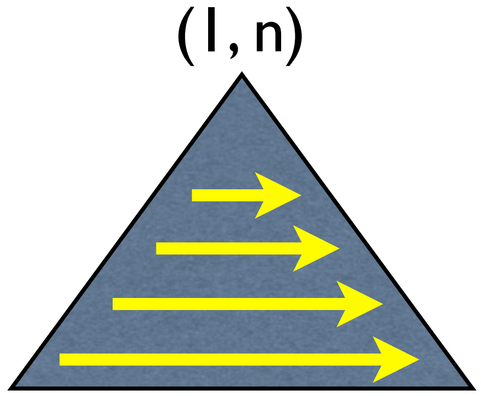


$$d(v) = \oplus_{(\langle u_1,u_2\rangle,v)\in E} [d(u_1) \otimes d(u_2) \otimes w(\langle u_1, u_2\rangle, v)]$$

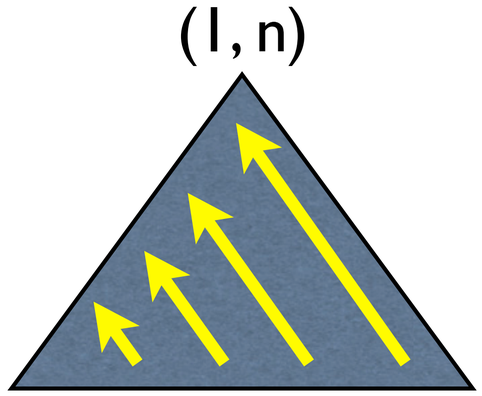
Example: RNA Folding and CKY Parsing

- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering
- Nussinov algorithm in RNA is almost identical to CKY but w/o overcounting

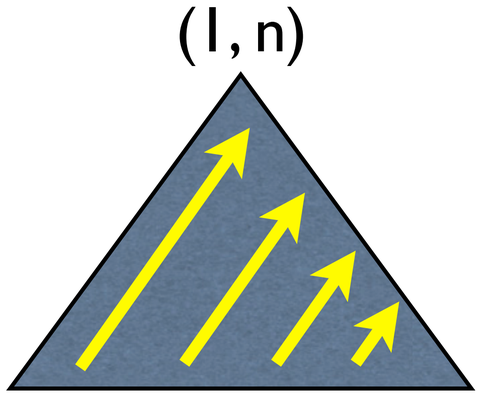




bottom-up



left-to-right

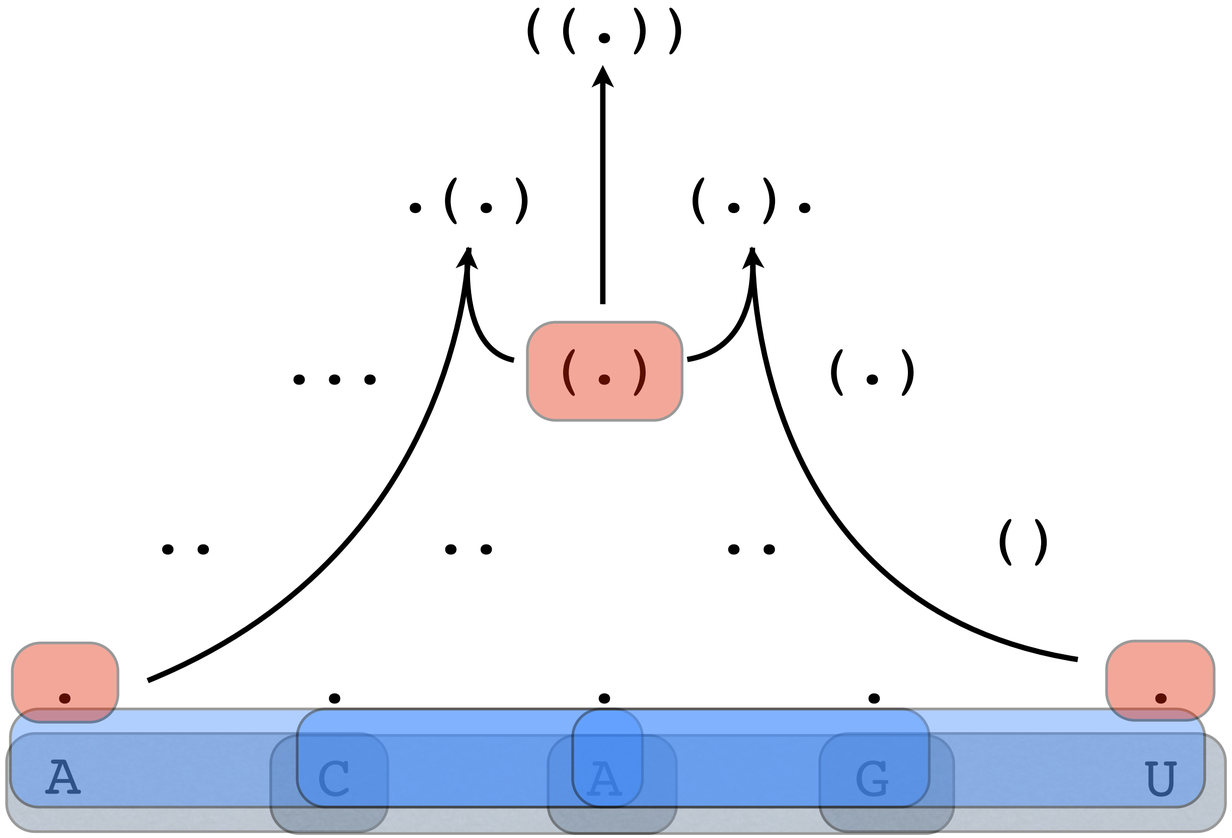


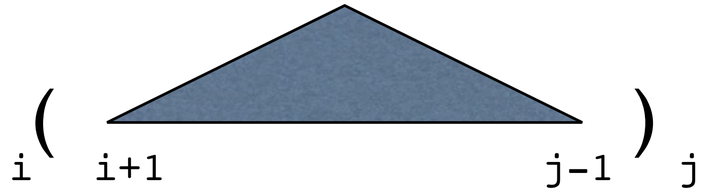
right-to-left

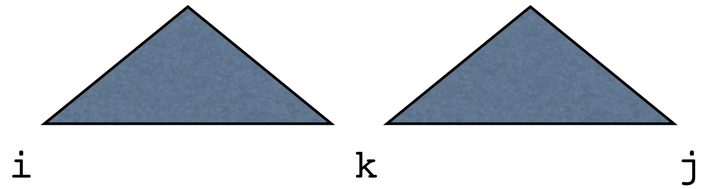
all $O(n^3)$

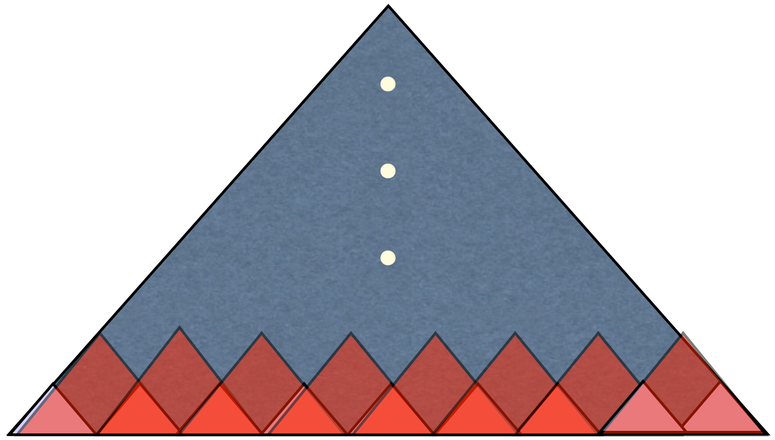
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
  - bottom-up CKY parsing
  - example: maximize # of pairs (A-U, G-C, or G-U)

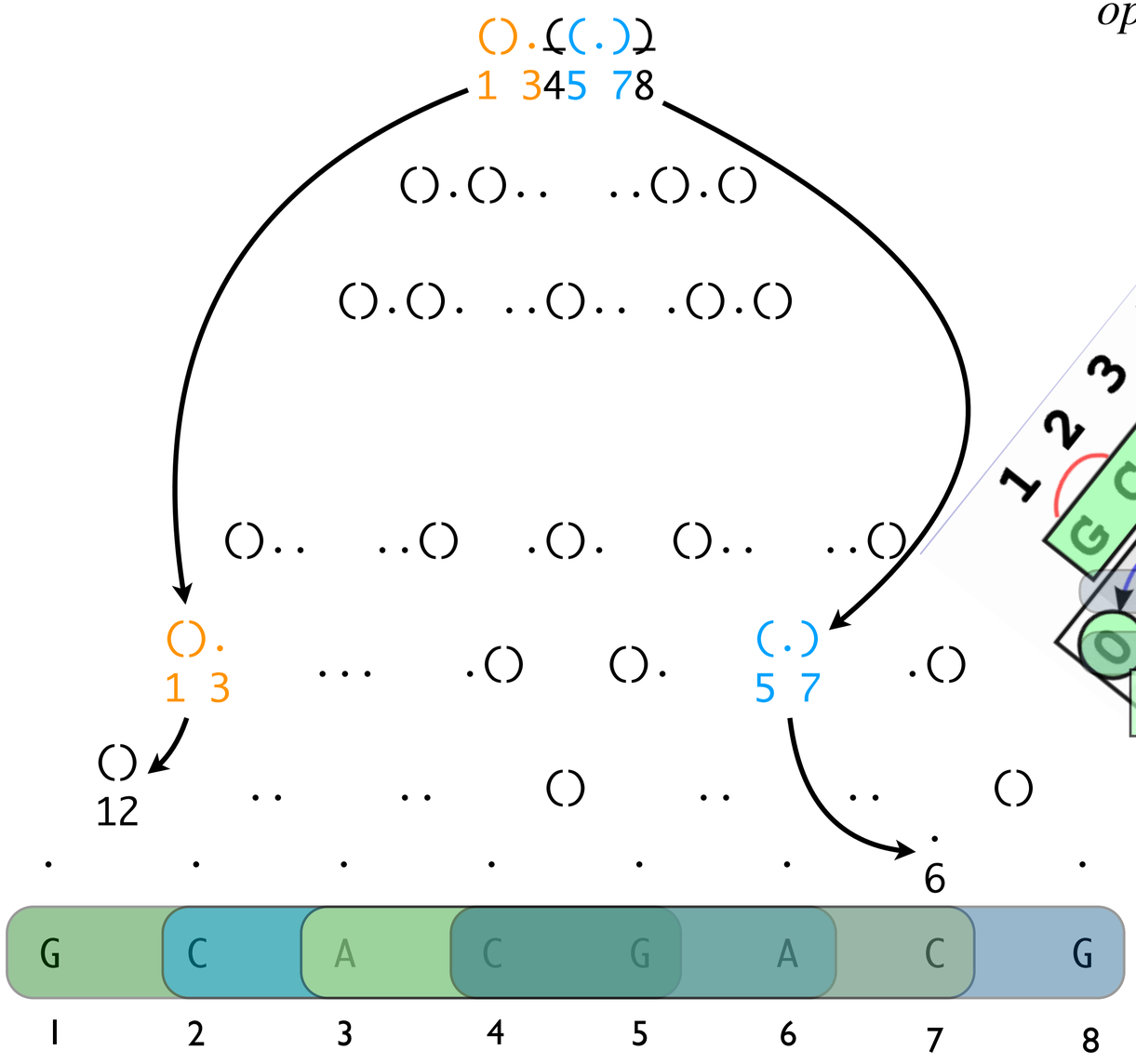


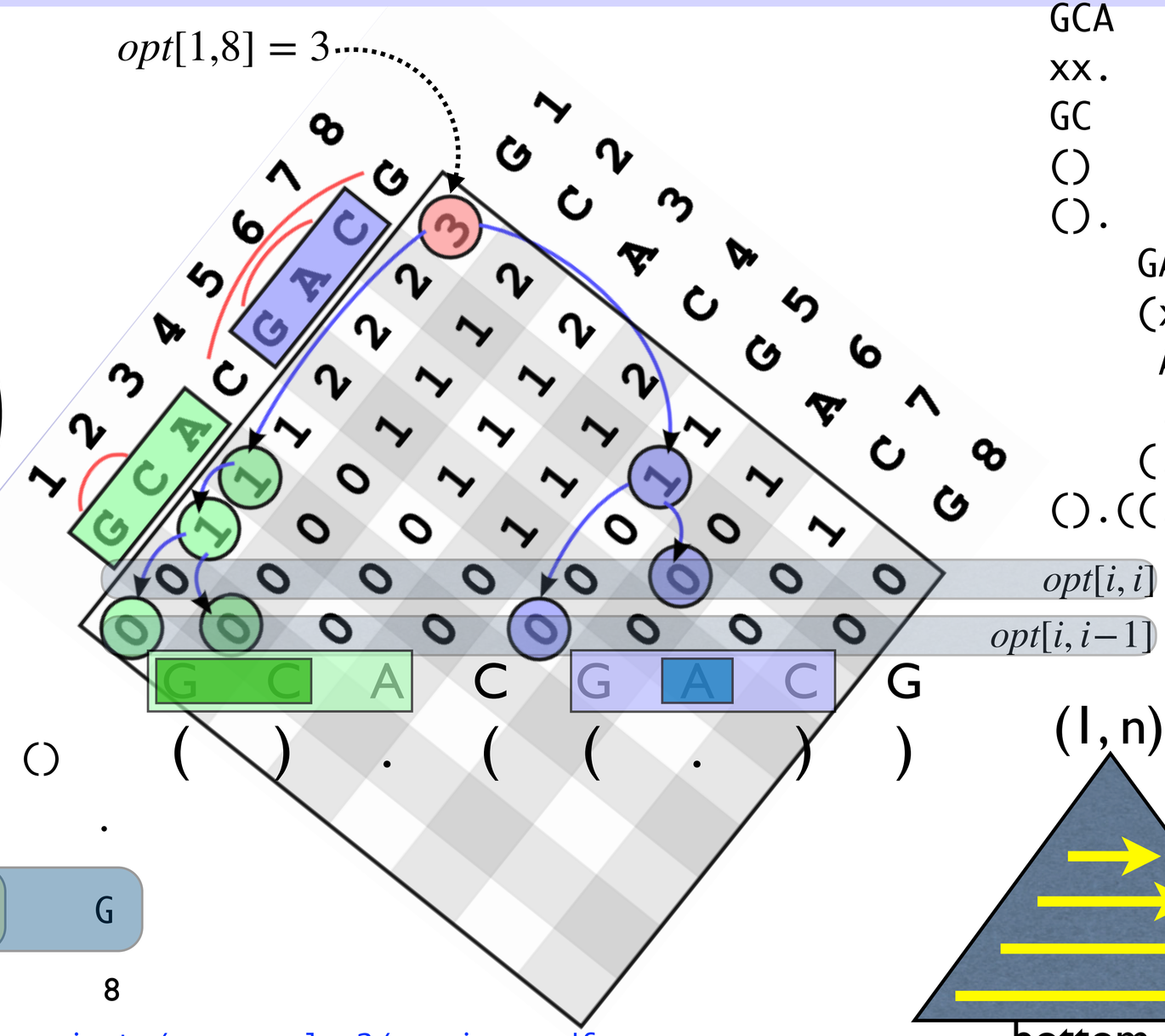




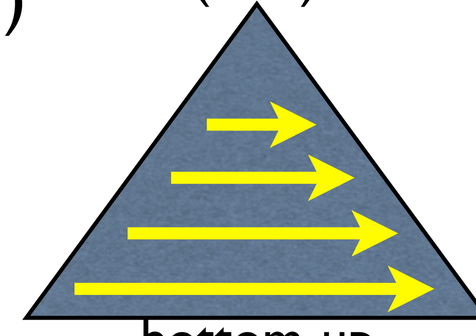


RNA Folding Example: Nussinov






```
12345678
GCACGACG
xxx(xxx)
GCA
xx.
GC
()
().
    GAC
    (x)
     A
     .
    (.)
().((.))
```



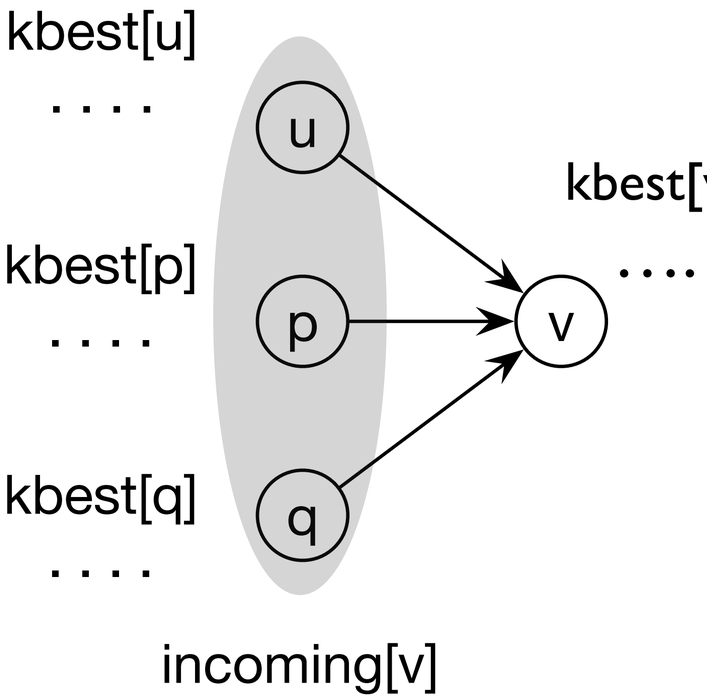
https://ad-publications.cs.uni-freiburg.de/student-projects/rna-google-2/nussinov.pdf

From 1-best to k-best

- each subproblem will now store top-k best answers instead of a single best
- we'll first extend Viterbi on DAGs to k-best Viterbi
- then extend generalized Viterbi on DAHs (e.g., CKY or Nussinov) to k-best

k-best Viterbi on Graph

- simple extension of Viterbi to solve k-best on graphs and hyper graphs
  cf. teams problem in HW4



for each node v,
  compute its kbest distances
  from the kbest of each incoming node u

1-best: $O(E+V)$

k-best: $O(E+V(d_{\max}+k\log d_{\max}))=O(E+E+Vk\log d_{\max})$
$=O(E+Vk\log d_{\max})$ where $d_{\max}$ is the max in-degree

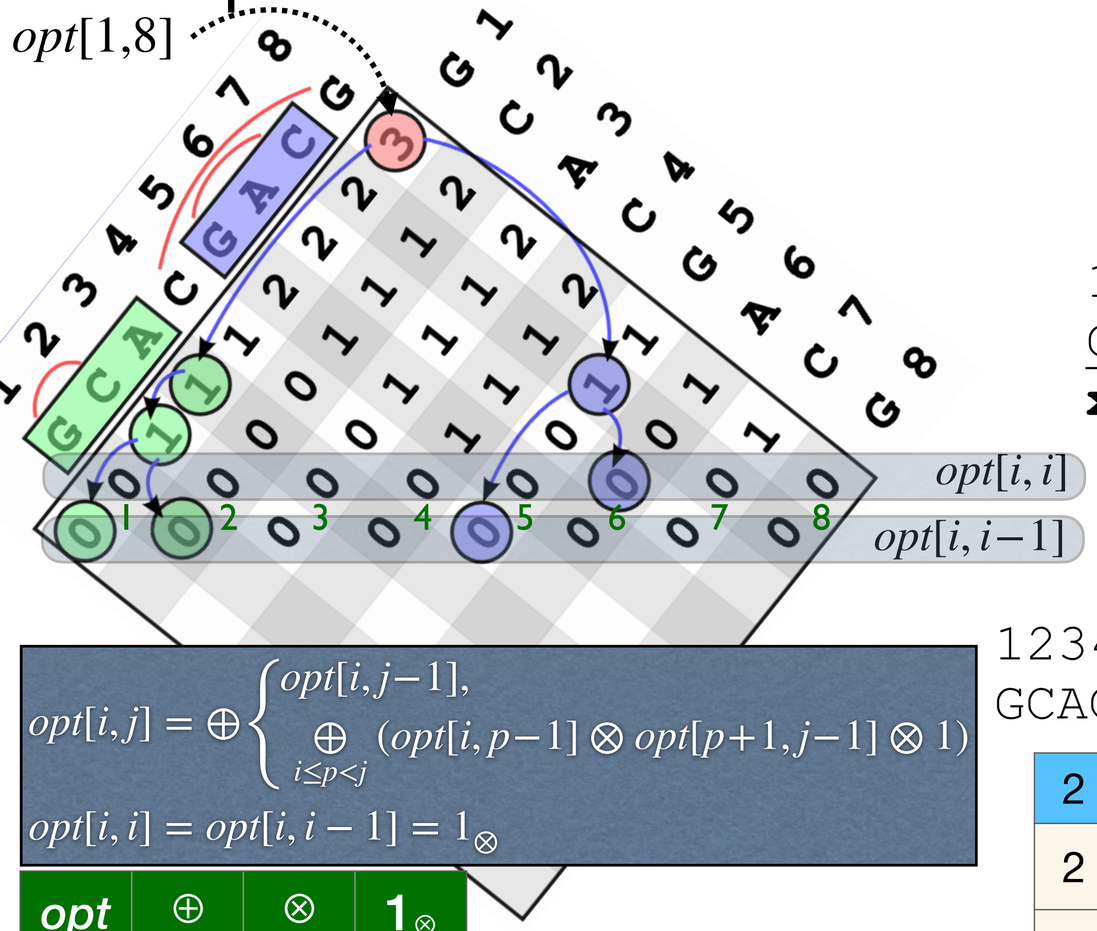
can improve it to: (cf. midterm & teams, w/ quickselect)

k-best: $O(E+Vk\log k)$ (assume $k \ll d_{\max}$)

("most states do not have anybody on team USA")

k-best Viterbi on Hypergraph

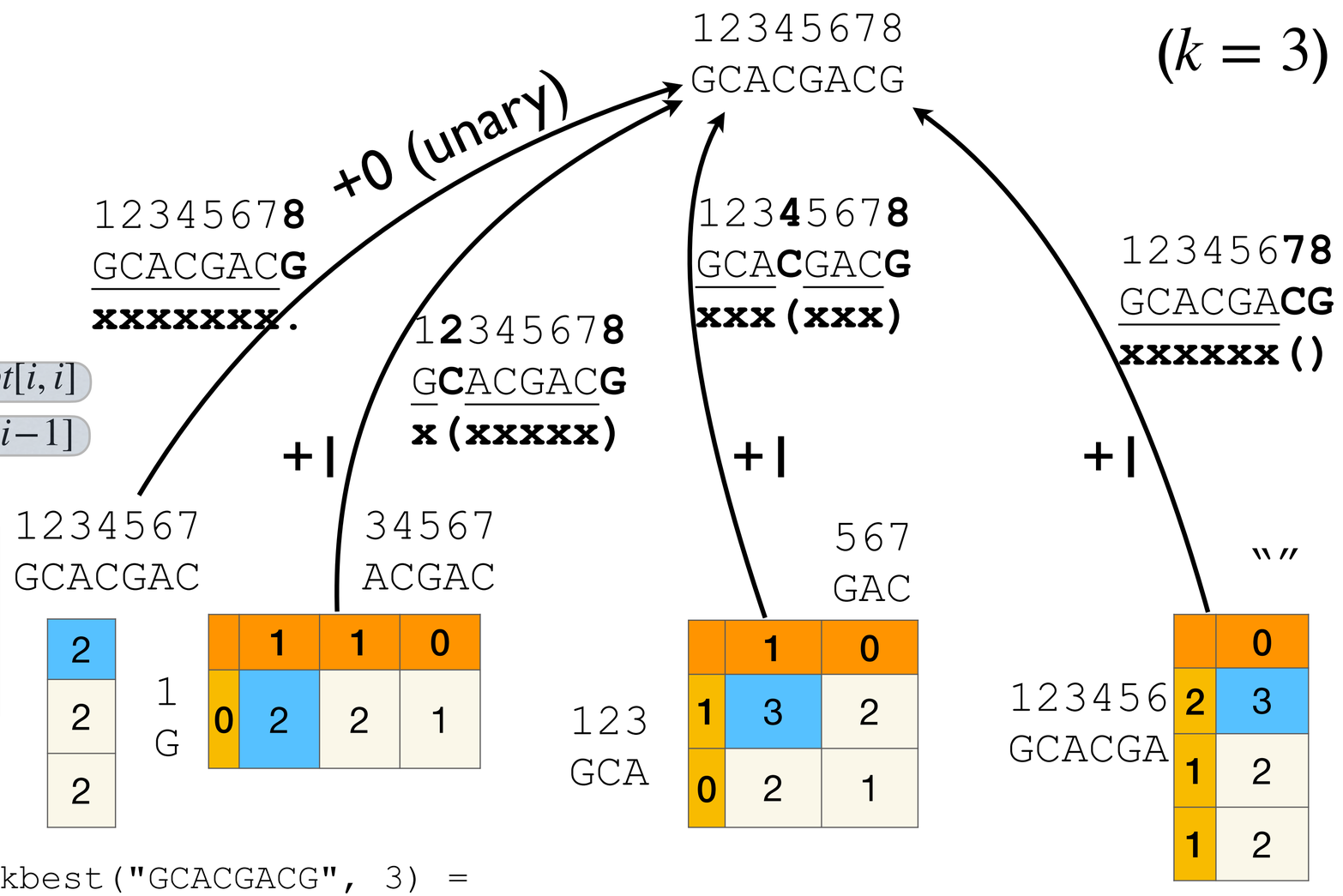
- simple extension of Viterbi to solve k-best on graphs and hyper graphs cf. midterm



$$opt[i,j] = \oplus \left\{ \begin{array}{l} opt[i,j-1], \\ \bigoplus_{i \le p < j} (opt[i,p-1] \otimes opt[p+1,j-1] \otimes 1) \end{array} \right.$$

$$opt[i,i] = opt[i,i-1] = 1_{\otimes}$$

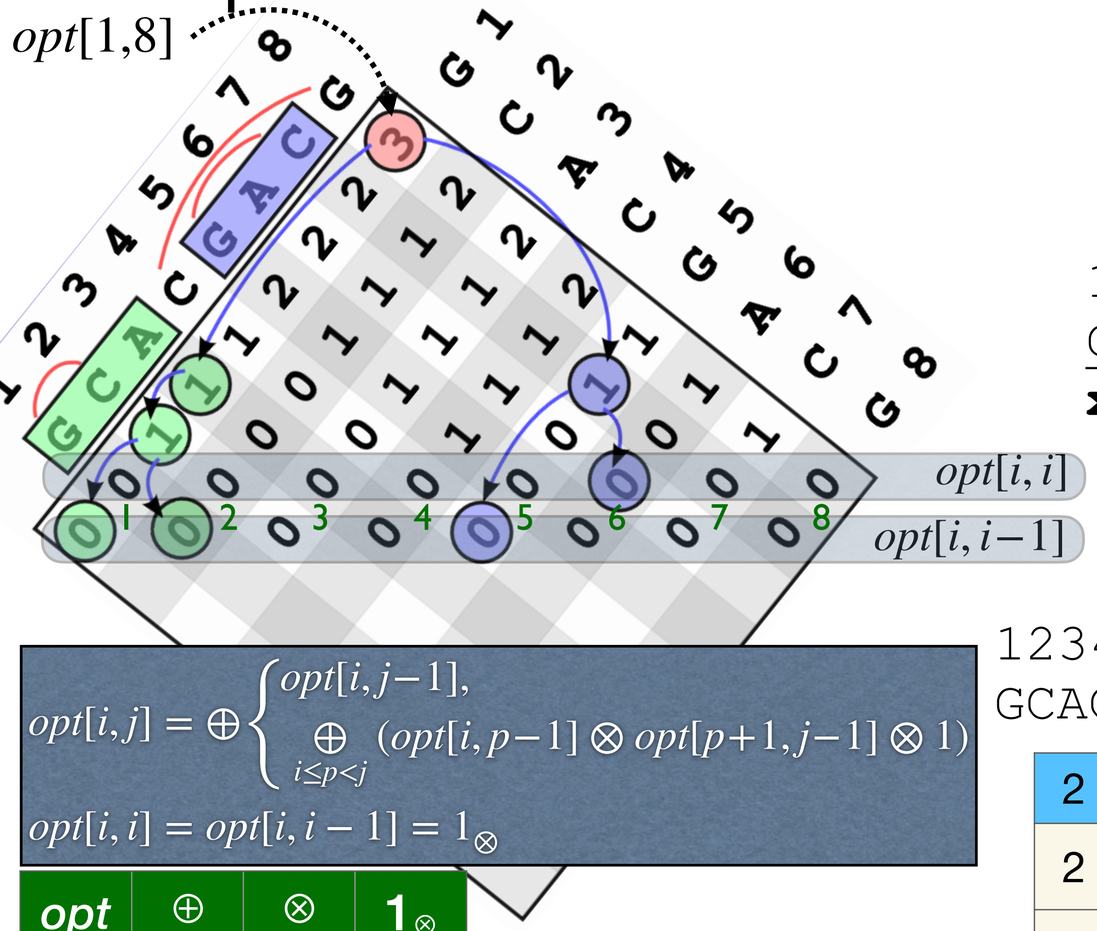
opt	⊕	⊗	$1_{\otimes}$
best	max	+	0
total	+	x	1




```
kbest("GCACGACG", 3) =
[]
```

k-best Viterbi on Hypergraph

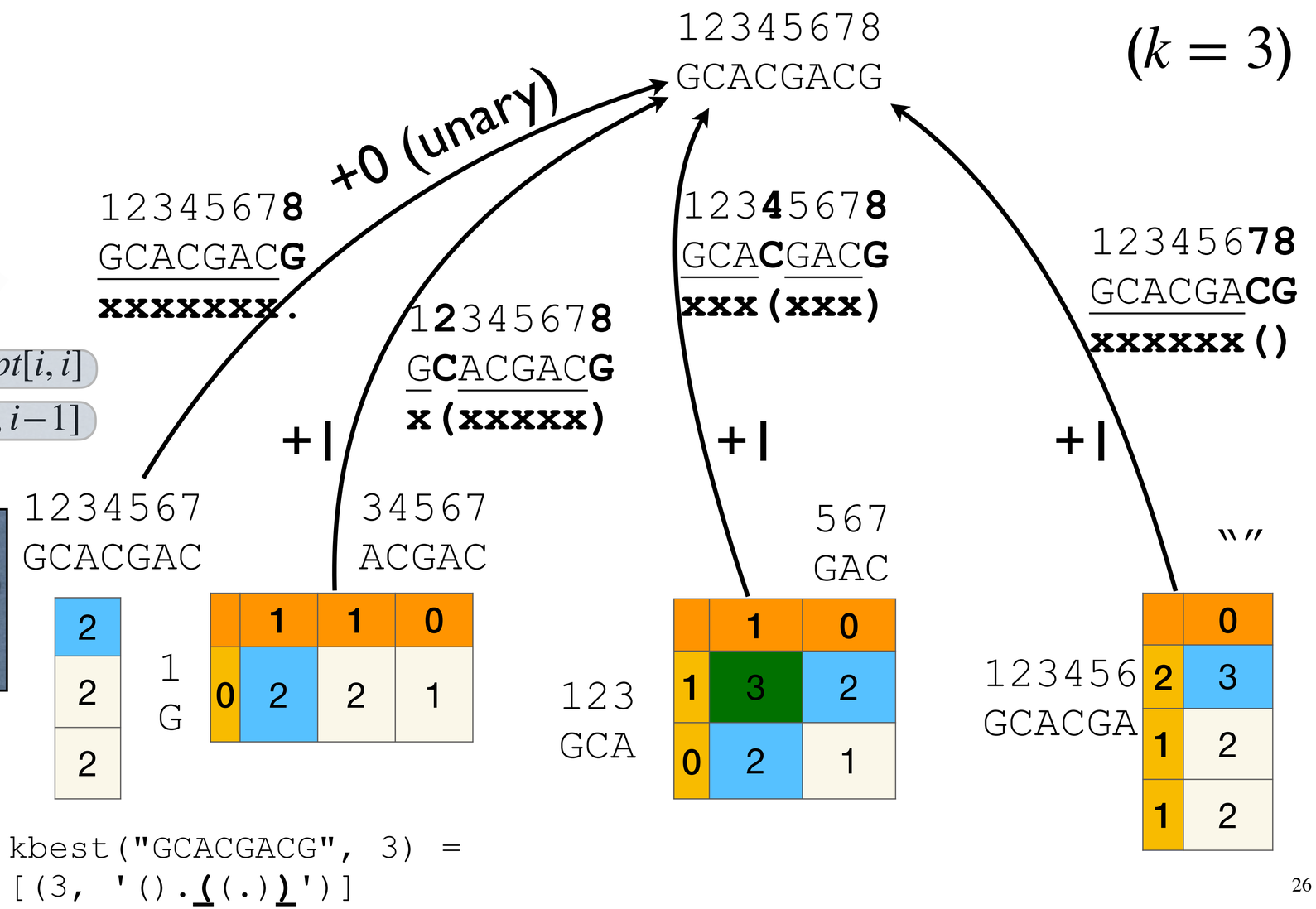
- simple extension of Viterbi to solve k-best on graphs and hyper graphs cf. midterm



$$opt[i,j] = \oplus \begin{cases} opt[i,j-1], \\ \bigoplus_{i \le p < j} (opt[i,p-1] \otimes opt[p+1,j-1] \otimes 1) \end{cases}$$

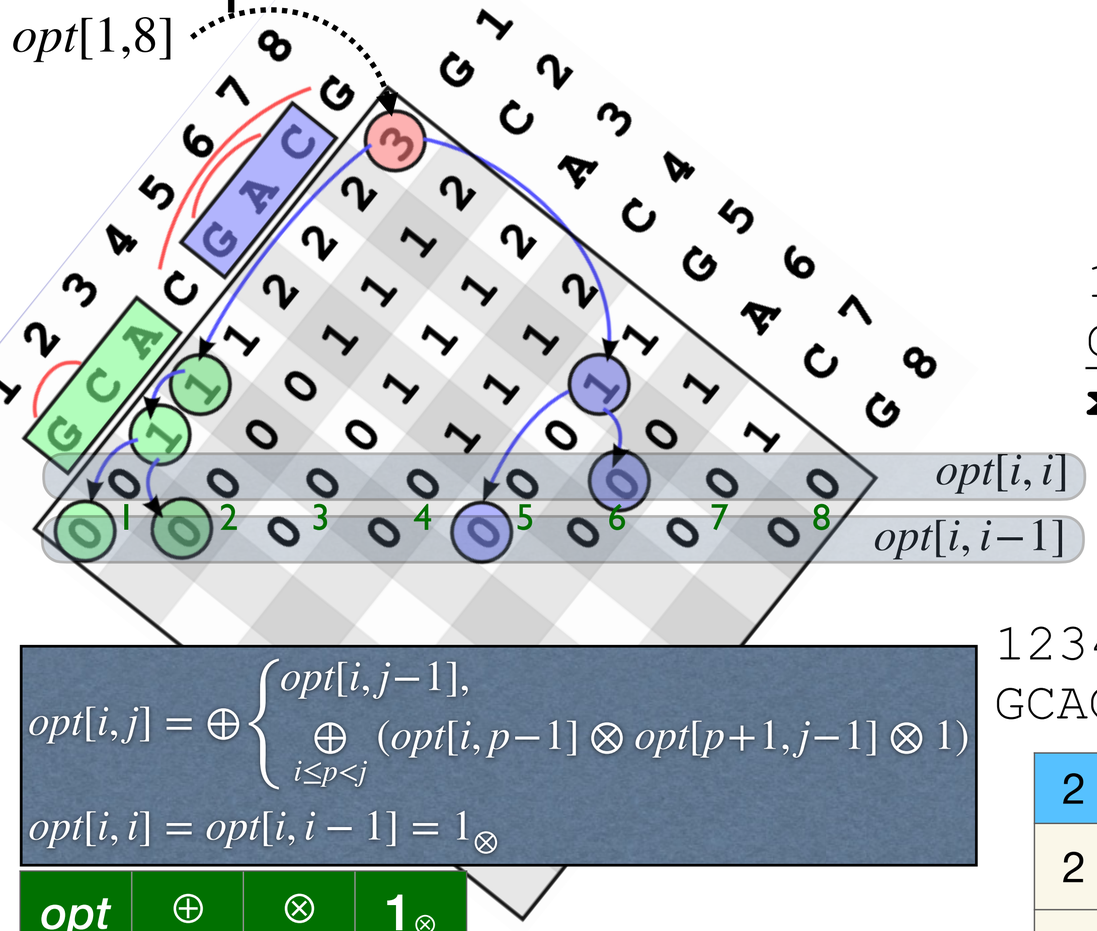
$$opt[i,i] = opt[i,i-1] = 1_{\otimes}$$

opt	⊕	⊗	$1_{\otimes}$
best	max	+	0
total	+	x	1



k-best Viterbi on Hypergraph

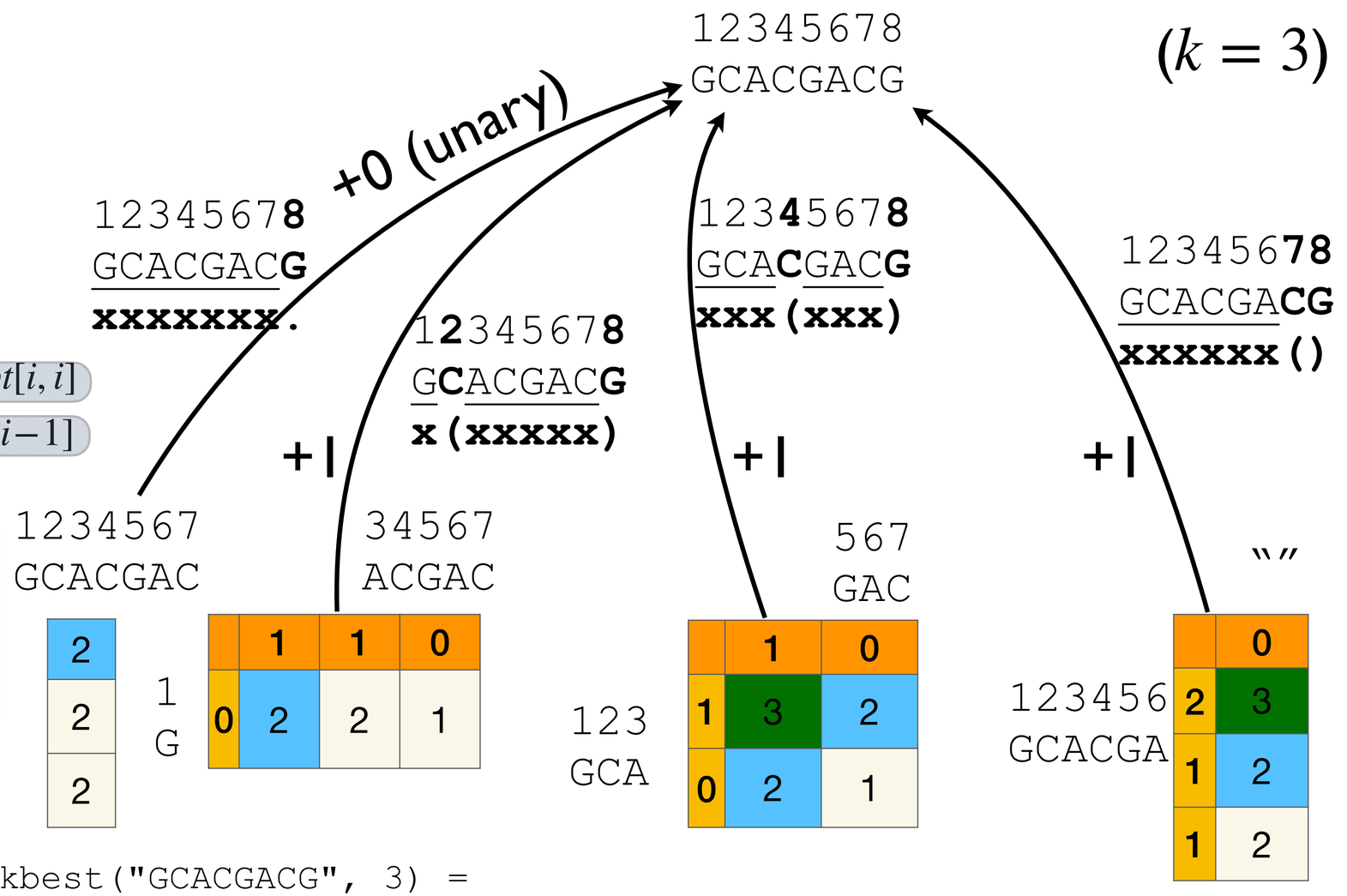
- simple extension of Viterbi to solve k-best on graphs and hyper graphs cf. midterm



$$opt[i,j] = \oplus \begin{cases} opt[i,j-1], \\ \bigoplus_{i \le p < j} (opt[i,p-1] \otimes opt[p+1,j-1] \otimes 1) \end{cases}$$

$$opt[i,i] = opt[i,i-1] = 1_{\otimes}$$

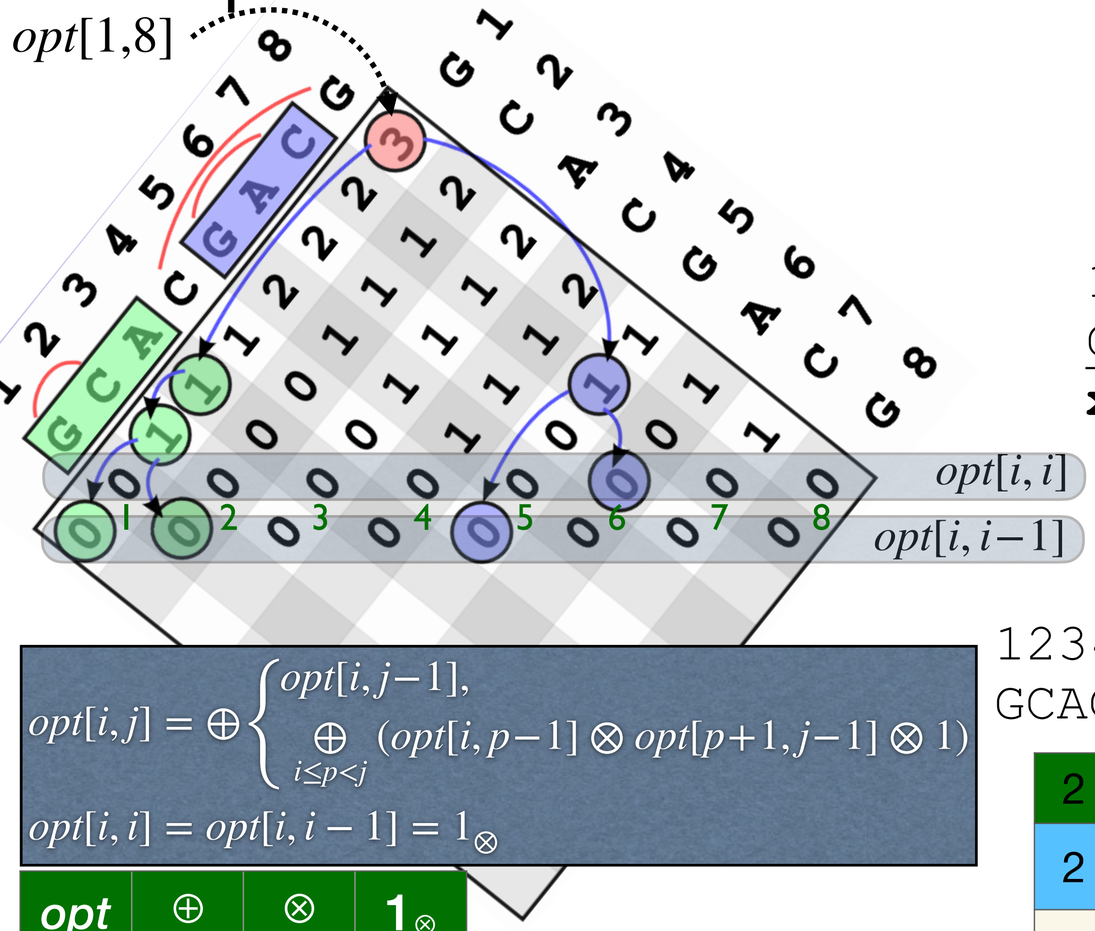
opt	⊕	⊗	$1_{\otimes}$
best	max	+	0
total	+	x	1




```
kbest("GCACGACG", 3) =
[(3, '().((.))'), (3, '().().()')]
```

k-best Viterbi on Hypergraph

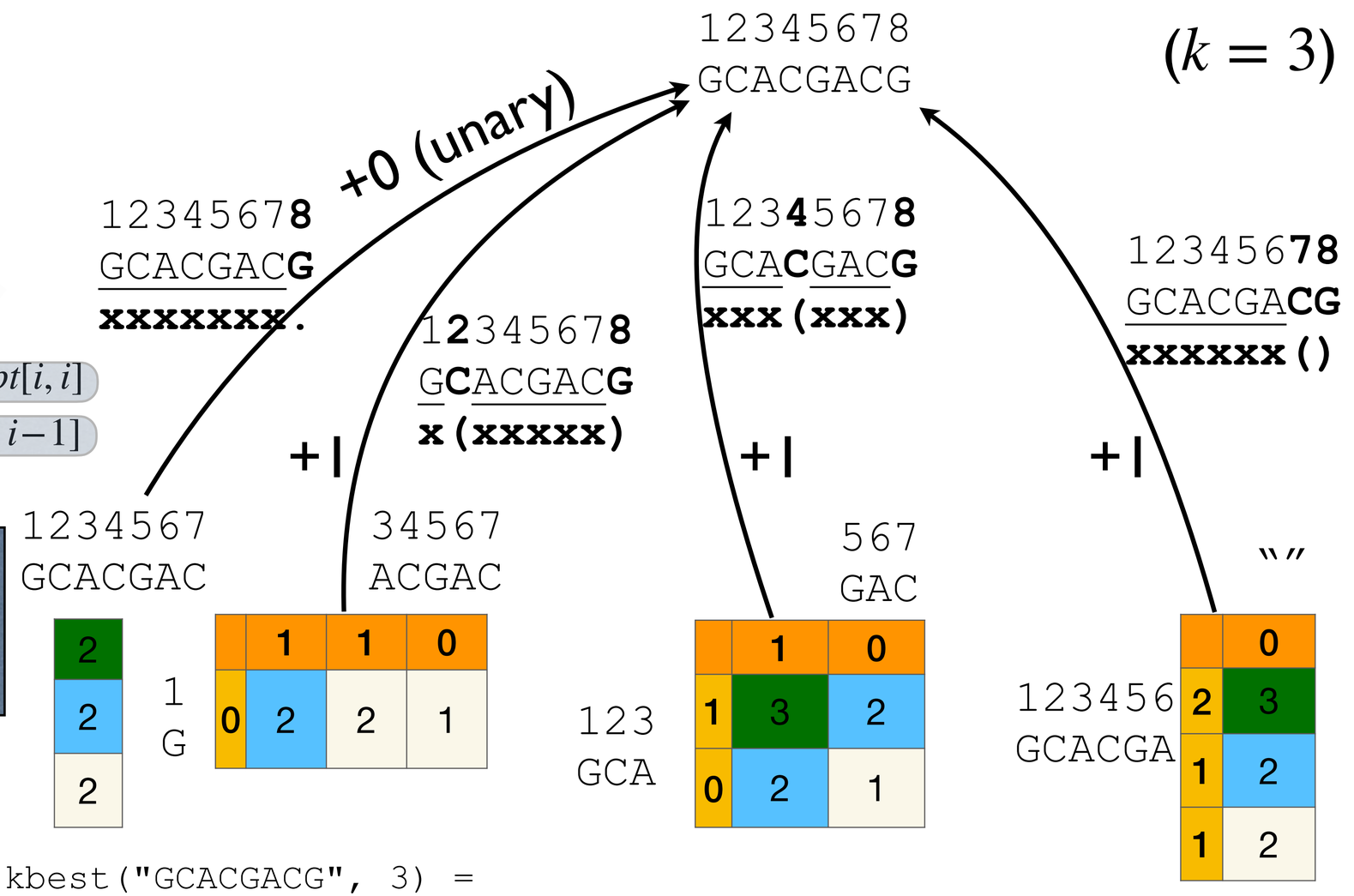
- simple extension of Viterbi to solve k-best on graphs and hyper graphs cf. midterm



$$opt[i,j] = \oplus \begin{cases} opt[i,j-1], \\ \bigoplus_{i \le p < j} (opt[i,p-1] \otimes opt[p+1,j-1] \otimes 1) \end{cases}$$

$$opt[i,i] = opt[i,i-1] = 1_{\otimes}$$

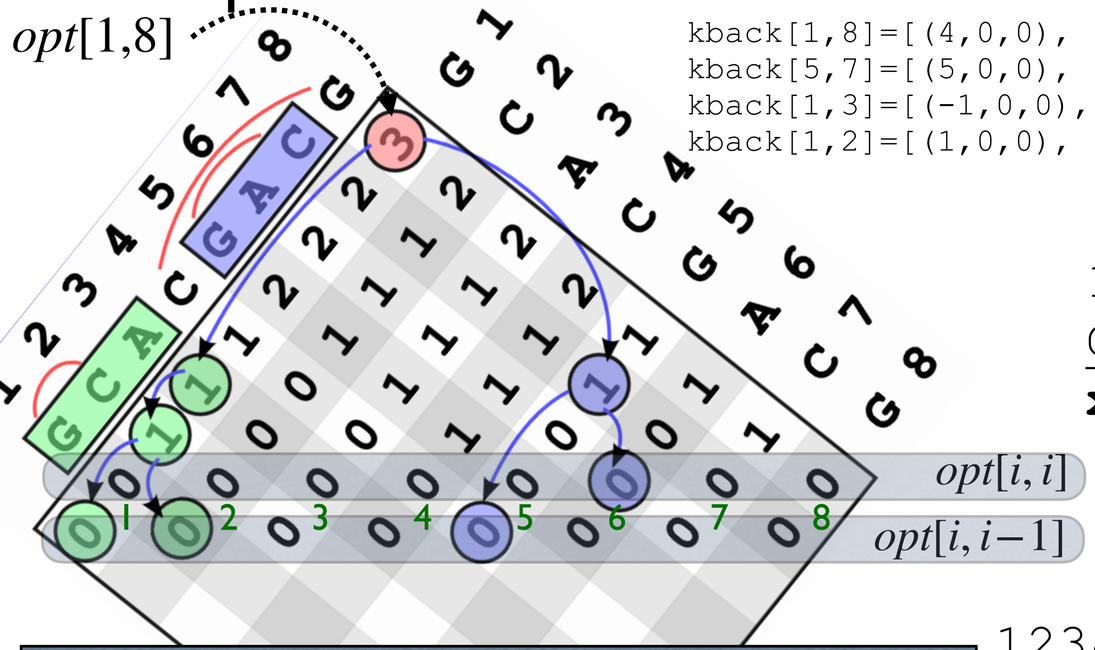
opt	⊕	⊗	$1_{\otimes}$
best	max	+	0
total	+	x	1




```
kbest("GCACGACG", 3) =
[(3, '().((.))'), (3, '().().()'), (2, '().()._')]
```

k-best Viterbi on Hypergraph

- simple extension of Viterbi to solve k-best on graphs and hyper graphs cf. midterm

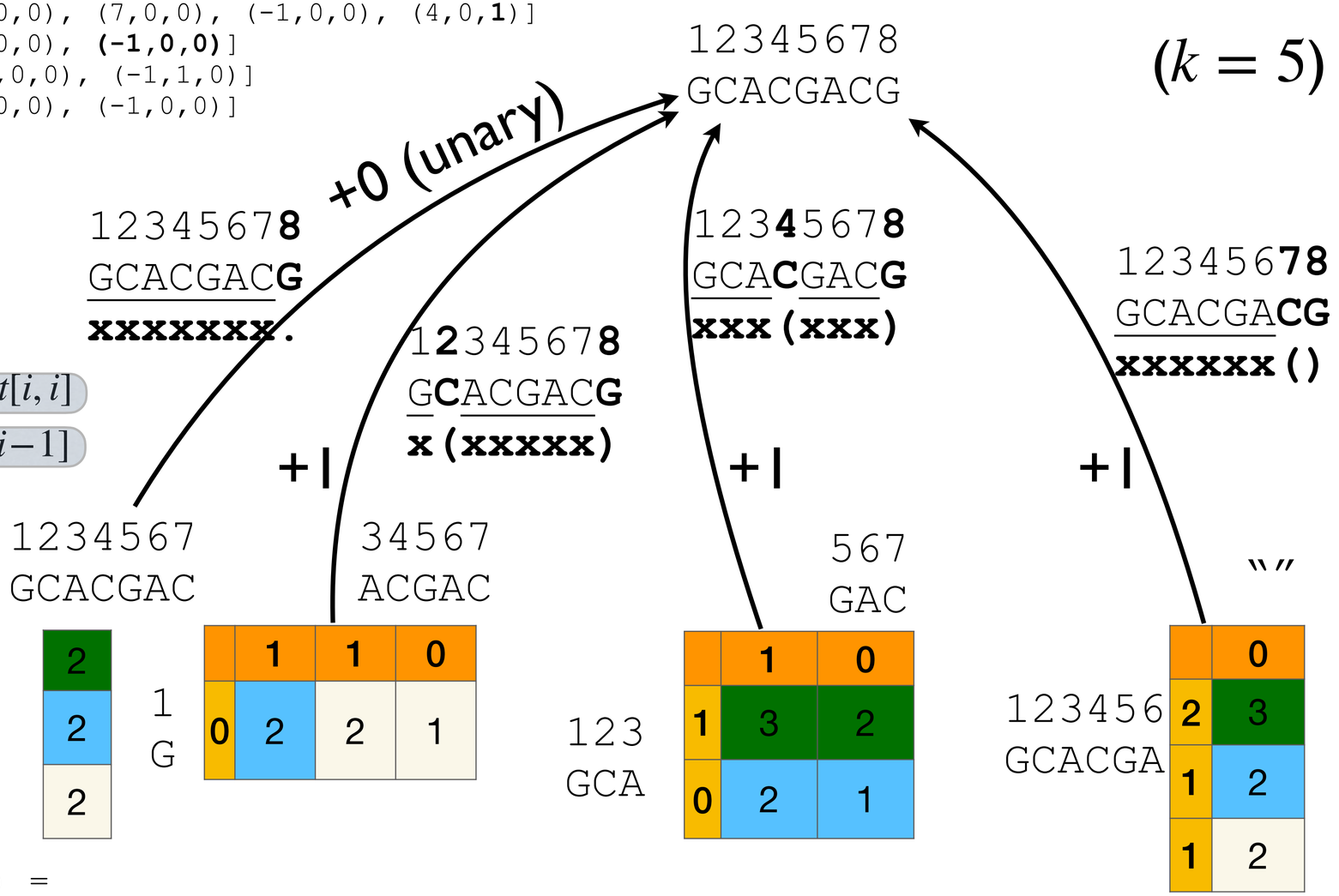



```
kback[1,8]=[(4,0,0), (7,0,0), (-1,0,0), (4,0,1)]
kback[5,7]=[(5,0,0), (-1,0,0)]
kback[1,3]=[(-1,0,0), (-1,1,0)]
kback[1,2]=[(1,0,0), (-1,0,0)]
```

$$opt[i,j] = \oplus \begin{cases} opt[i,j-1], \\ \bigoplus_{i \le p < j} (opt[i,p-1] \otimes opt[p+1,j-1] \otimes 1) \end{cases}$$

$$opt[i,i] = opt[i,i-1] = 1_{\otimes}$$

opt	⊕	⊗	$1_{\otimes}$
best	max	+	0
total	+	x	1

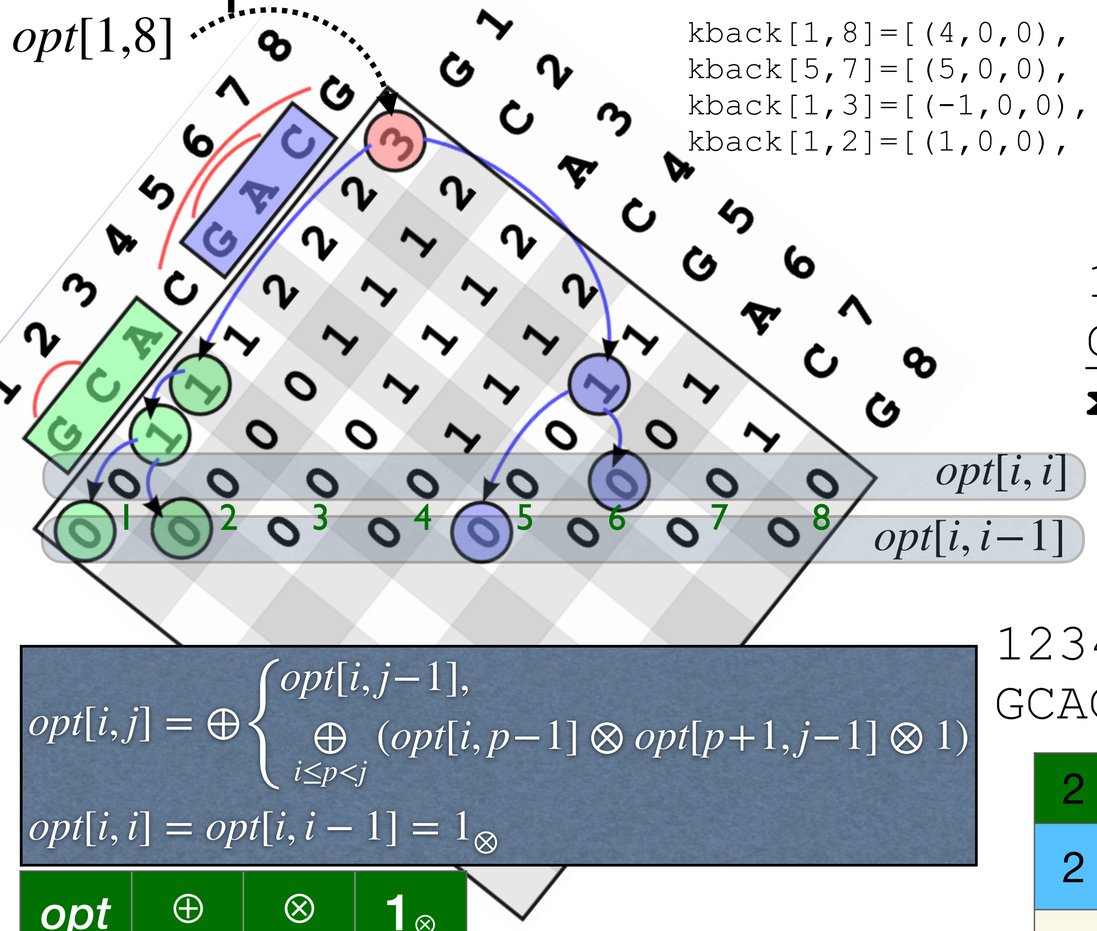



```
kbest("GCACGACG", 5) =
[(3, '().((.))'), (3, '().().()'), (2, '().()...'), (2, '().(...)')]
```

k-best Viterbi on Hypergraph

```
12345678 backtrace:
GCACGACG sol(1,8,4)
xxx(xxx)    [4,1,0]
GCA      sol(1,3,1)
xx.        [-1,1,0]
GC       sol(1,2,1)
x.         [-1,0,0]
G        sol(1,1,0)
.
    GAC  sol(5,7,0)
    (x)     [5,0,0]
     A   sol(6,6,0)
     .
...((.)) <-5th-best
```

- simple extension of Viterbi to solve k-best on graphs and hyper graphs

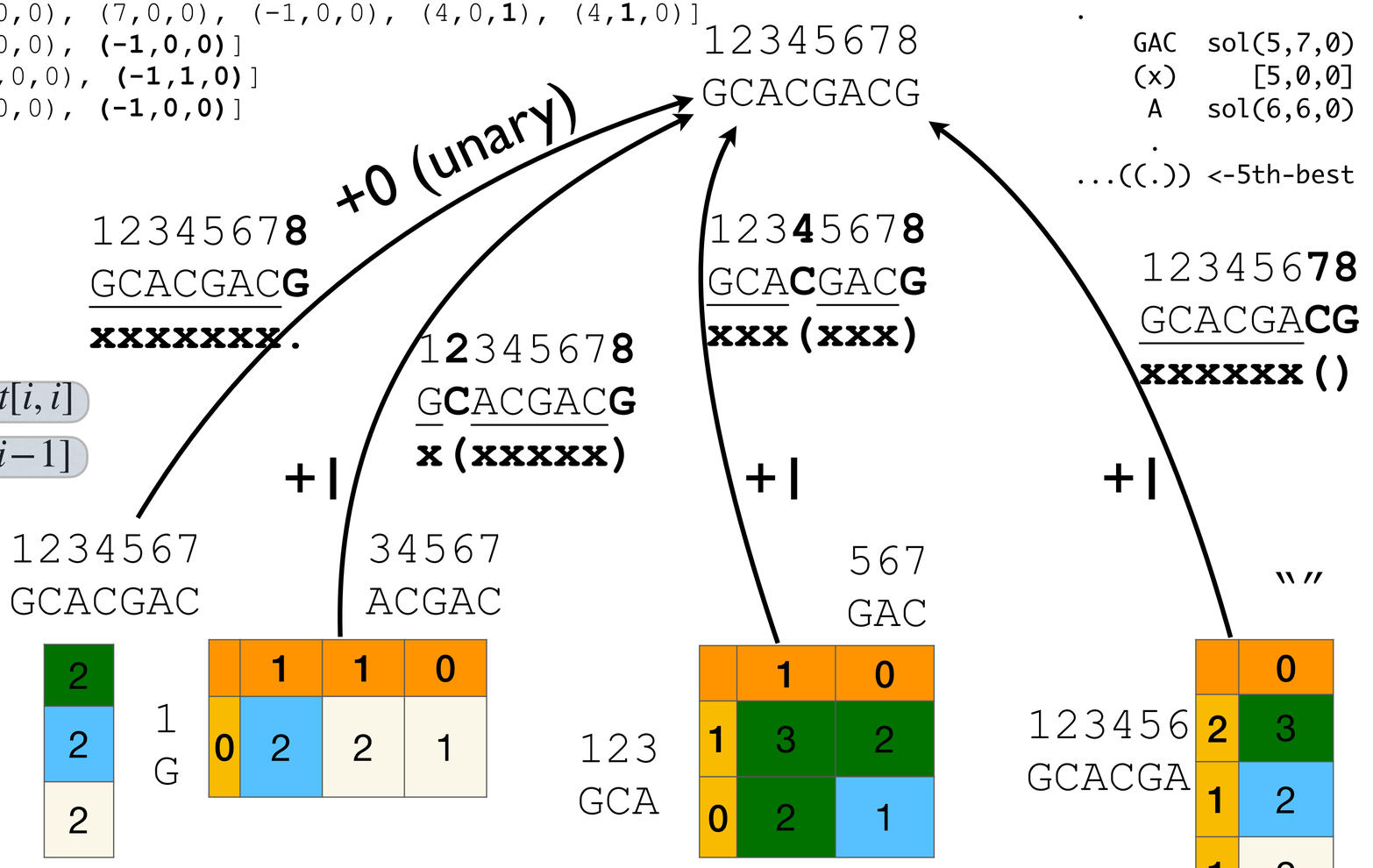



```
kback[1,8]=[(4,0,0), (7,0,0), (-1,0,0), (4,0,1), (4,1,0)]
kback[5,7]=[(5,0,0), (-1,0,0)]
kback[1,3]=[(-1,0,0), (-1,1,0)]
kback[1,2]=[(1,0,0), (-1,0,0)]
```

$$opt[i,j] = \oplus \begin{cases} opt[i,j-1], \\ \bigoplus_{i \le p < j} (opt[i,p-1] \otimes opt[p+1,j-1] \otimes 1) \end{cases}$$

$$opt[i,i] = opt[i,i-1] = 1_{\otimes}$$

opt	⊕	⊗	$1_{\otimes}$
best	max	+	0
total	+	x	1




```
kbest("GCACGACG", 5) =
[(3, '().((.))'), (3, '().().()'), (2, '().()...'), (2, '().(...)'), (2, '...((.))')]
```