How Do We Compare Biological Sequences?

Dynamic Programming and Divide-and Conquer Algorithms

Phillip Compeau and Pavel Pevzner.

Bioinformatics Algorithms: An Active Learning Approach

How Do We Compare Biological Sequences

- **From Sequence Comparison to Biological Insights**
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

RNA Tie Club

ribosome

DNA → RNA → Protein

genetic code

RNA tie club

From Genetic Code to Non-Ribosomal Code

non-ribosomal peptide synthetase

ribosome

DNA → RNA → Protein → Peptide

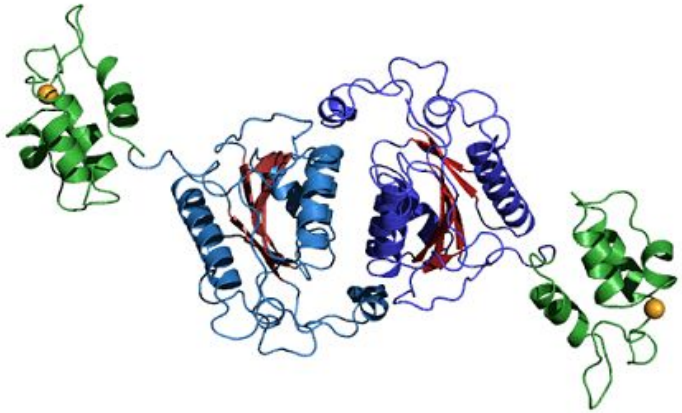
genetic code

non-ribosomal code

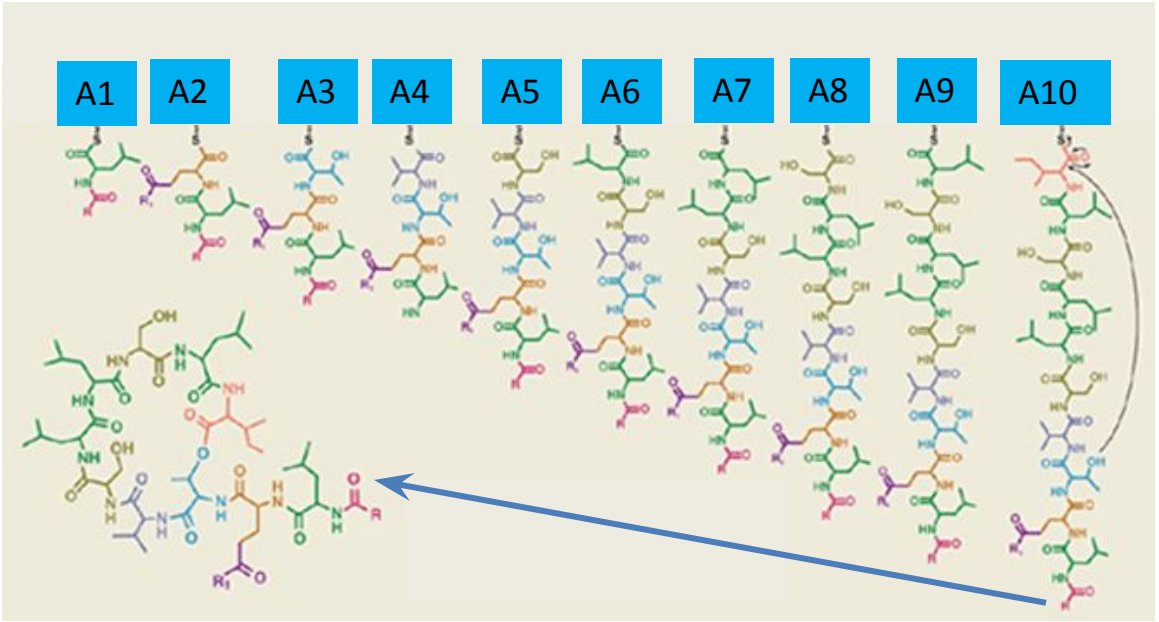
RNA tie club

Marahiel

NRP Synthetase: A Giant Molecular Assembly Line



Adenylation domains



NRP synthetase adds one amino acid at a time

These Three A-domains Do Not Look Similar

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA

AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS

These Three A-domains Do Not Look Similar

YAFDLGYTCMFPV**L**LG**G**GELHIVQKETYTAPDEIAHYI**K**EHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA

AFDVSAGDFARAL**L**TG**G**QLIVCPNEVKMDPASLYAIIK**K**YDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAP**L**LN**G**GTVVCIDYYTTIDIKALEAVF**K**QHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS

just 3 conservative columns

Do they look similar now?

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA

-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS
```

11 conservative columns

And now?

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA

-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS

19 conservative columns!

Red Positions Encode Conservative Core of A-domains

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Which positions are responsible for encoding **different** amino acids Asp, Orn, Val?

Blue Positions in A-domains Define Non-Ribosomal Code

YAFD**L**GY**T**CMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYI**K**LTPSLFHTIVNTASFAFDANFESLRLI**V**L**G**GEKIIPIDVIAFRKMYGHTE-FIN**H**YGPTEAT**IG**A

-AFD**V**SA**G**DFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIF**E**ATPALVIPLMEYI-YEQKLDISQLQIL**I**V**G**SDSCSME**D**FKTLVSRFGSTIRIVN**S**YGVTEAC**ID**S

IAFD**A**SS**W**EIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGA**M**LPPALLKQCLVSA----PTMISSLEIL**F**A**A**GDRLSSQDAILARRAVGSGV-Y-N**A**YGPTENT**VL**S

LTKVGHIG	**Asp**
VGEIGSID	**Orn**
AWMFAAVL	**Val**

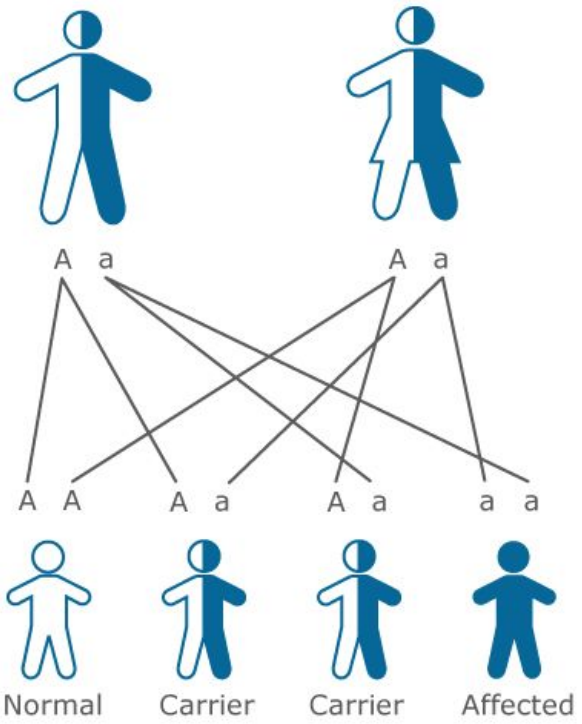
Cystic Fibrosis

- **Cystic fibrosis (CF)**: An often fatal disease which affects the respiratory system and produces an abnormally large amount of mucus.
  - Mucus is a slimy material that coats epithelial surfaces and is secreted into fluids such as saliva.

Approximately 1 in 25 Humans Carry a Faulty CF Gene

- In the early 1980s biologists hypothesized that CF is caused by mutations in an unidentified gene.

- When BOTH parent carry a faulty gene, there is a 25% chance that their child will have cystic fibrosis.



Where Is the Cystic Fibrosis Gene?

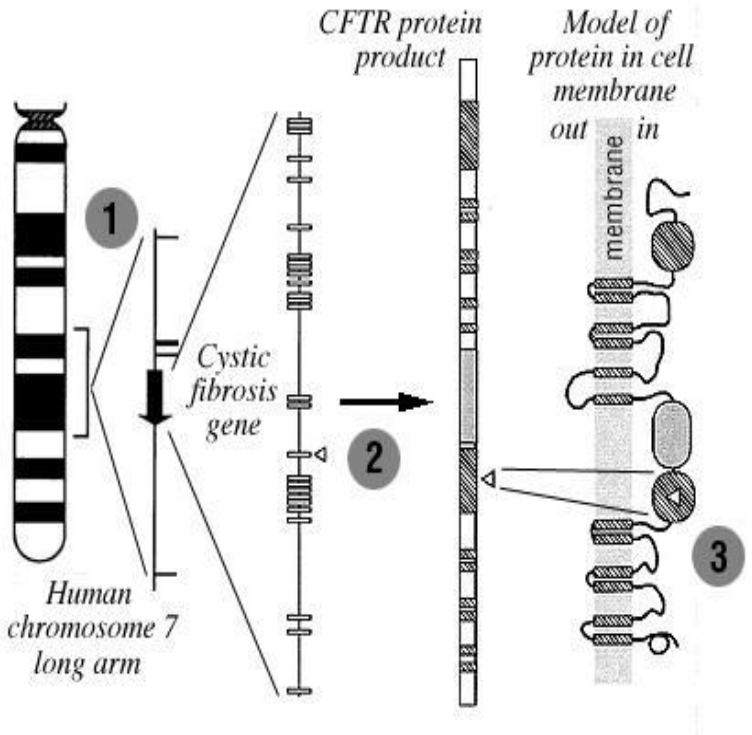
- In the late 1980s, biologists narrowed the search for the CF gene to a small region on chromosome 7.
- One of these genes was **similar** to **ATP binding proteins** that act as transport channels responsible for secretion.
- **Hint:** cystic fibrosis involves sweet secretion with abnormally high sodium levels.

Chromosome 7

Adenosine Triphosphate (ATP) transports chemical energy within cells.

Where Is the Cystic Fibrosis Gene?

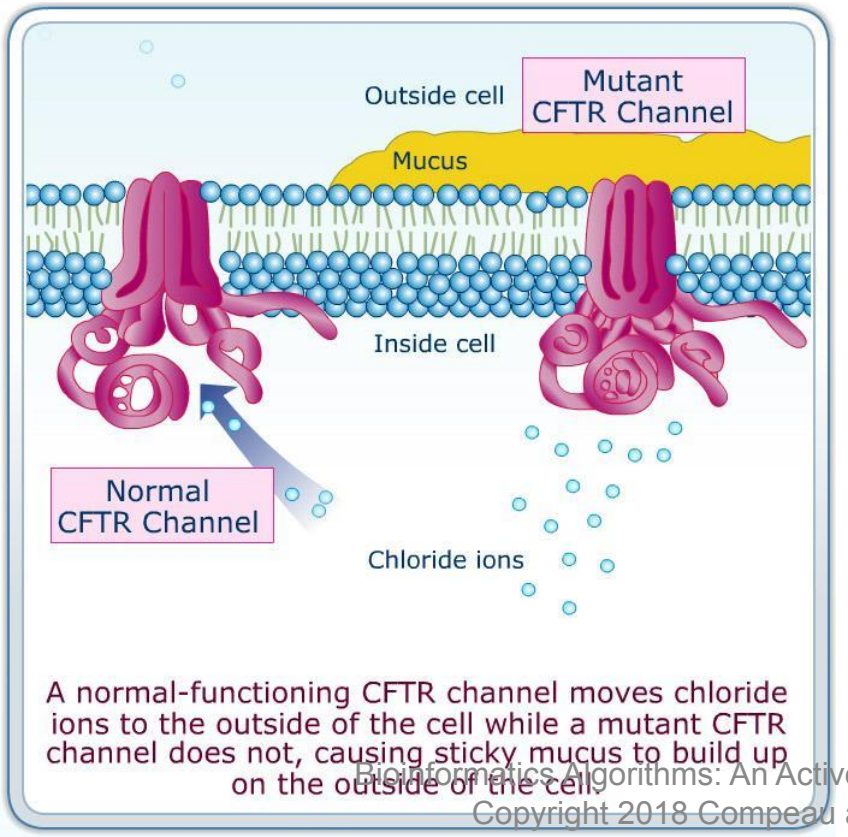
- In the late 1980s, biologists narrowed the search for the CF gene to a small region on chromosome 7.
- One of these genes was **similar** to **ATP binding proteins** that act as transport channels responsible for secretion.
- **Hint:** cystic fibrosis involves sweet secretion with abnormally high sodium levels.

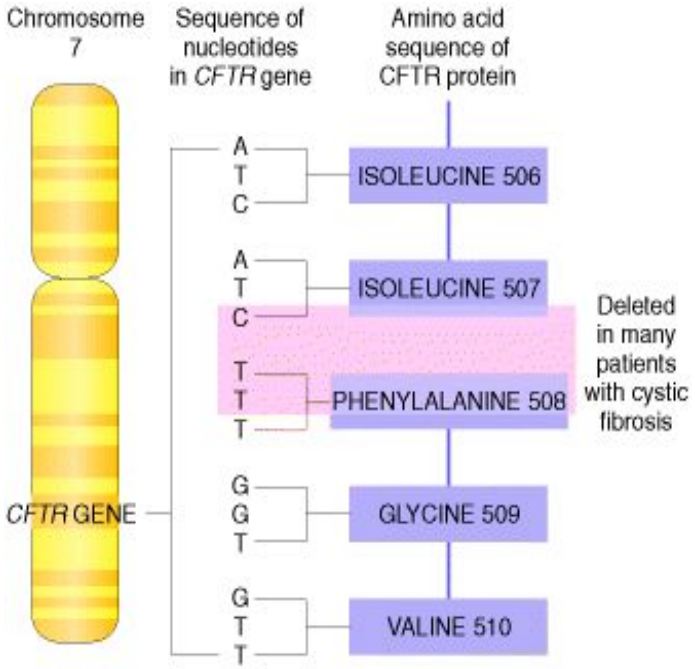


Adenosine Triphosphate (ATP) transports chemical energy within cells.

CFTR: Cystic Fibrosis Transmembrane Conductance Regulator

- The CFTR protein controls the flow of ions in and out of cells inside the lungs.





How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- **The Alignment Game and the Longest Common Subsequence**
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

The Alignment Game

```
A T G T T A T A
A T C G T C C
```

Alignment Game (maximizing the number of points):

- Remove the 1st symbol from each sequence
  - 1 point if the symbols match, 0 points if they don't match
- Remove the 1st symbol from one of the sequences
  - 0 points

The Alignment Game

```
 A T G T T A T A
 A T C G T C C
+1
```

The Alignment Game

```
 A T G T T A T A
 A T C G T C C
+1+1
```

The Alignment Game

```
 A T - G T T A T A
 A T C G T C C
+1+1
```

The Alignment Game

```
 A T - G T T A T A
 A T C G T C C
+1+1  +1
```

The Alignment Game

```
 A T - G T T A T A
 A T C G T C C
+1+1  +1+1
```

The Alignment Game

```
 A T - G T T A T A
 A T C G T - C C
+1+1  +1+1
```

The Alignment Game

```
 A T - G T T A T A
 A T C G T - C C
+1+1  +1+1
```

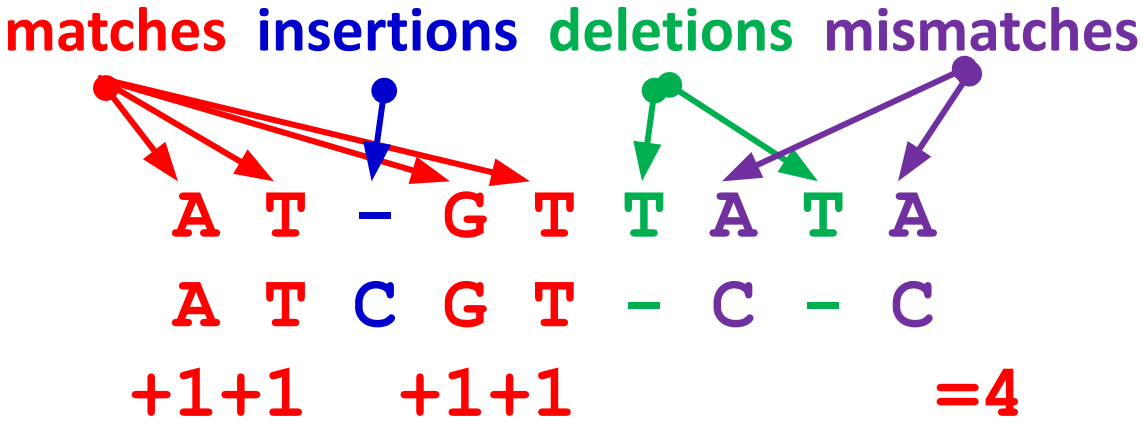
The Alignment Game

```
 A T - G T T A T A
 A T C G T - C - C
+1+1  +1+1
```

The Alignment Game

```
 A T - G T T A T A
 A T C G T - C - C
+1+1  +1+1       =4
```

What Is the Sequence Alignment?



Alignment of two sequences is a two-row matrix:

1st row: symbols of the 1st sequence (in order) interspersed by "-"
2nd row: symbols of the 2nd sequence (in order) interspersed by "-"

Longest Common Subsequence

```
A T - G T T A T A
A T C G T - C - C
```

Matches in alignment of two sequences (**ATGT**) form their **Common Subsequence**

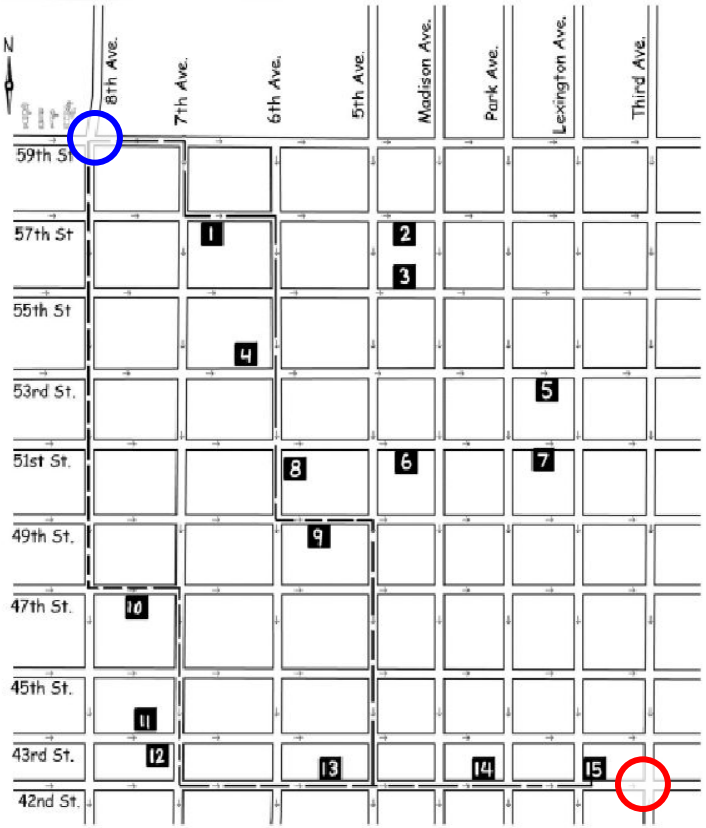
Longest Common Subsequence Problem: Find a longest common subsequence of two strings.

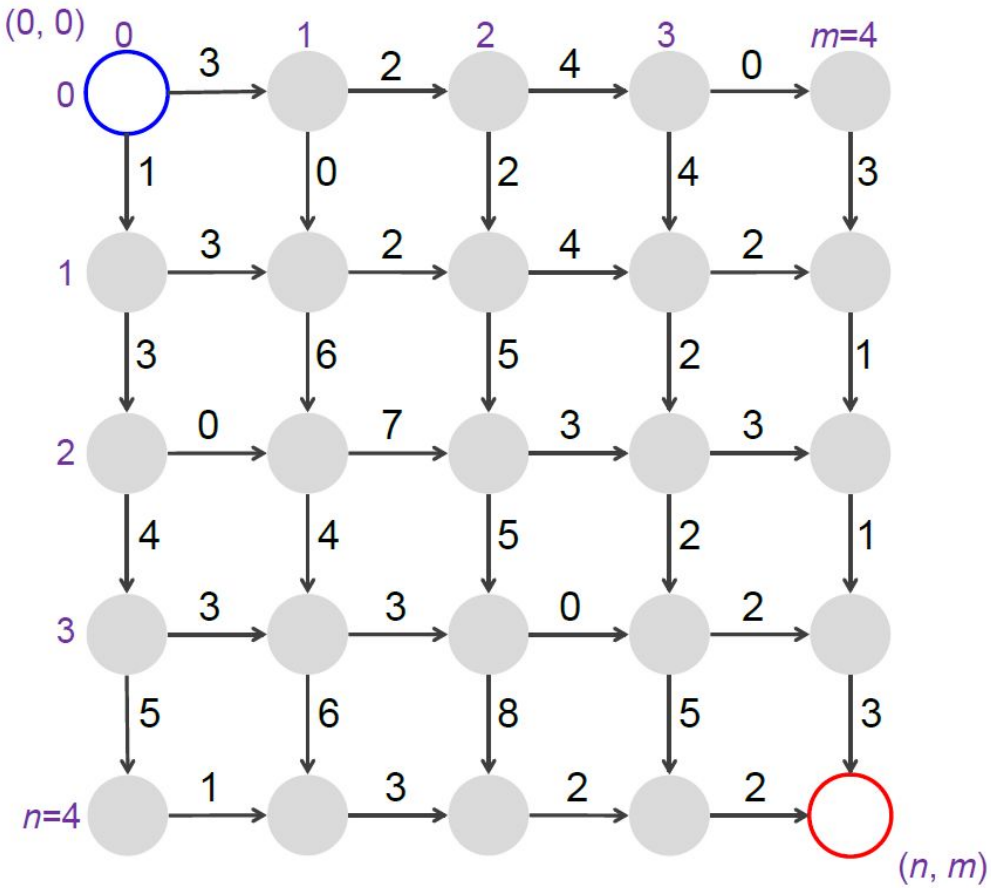
- **Input:** Two strings.
- **Output:** A longest common subsequence of these strings.

How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- **The Manhattan Tourist Problem**
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

From Manhattan to a Grid Graph



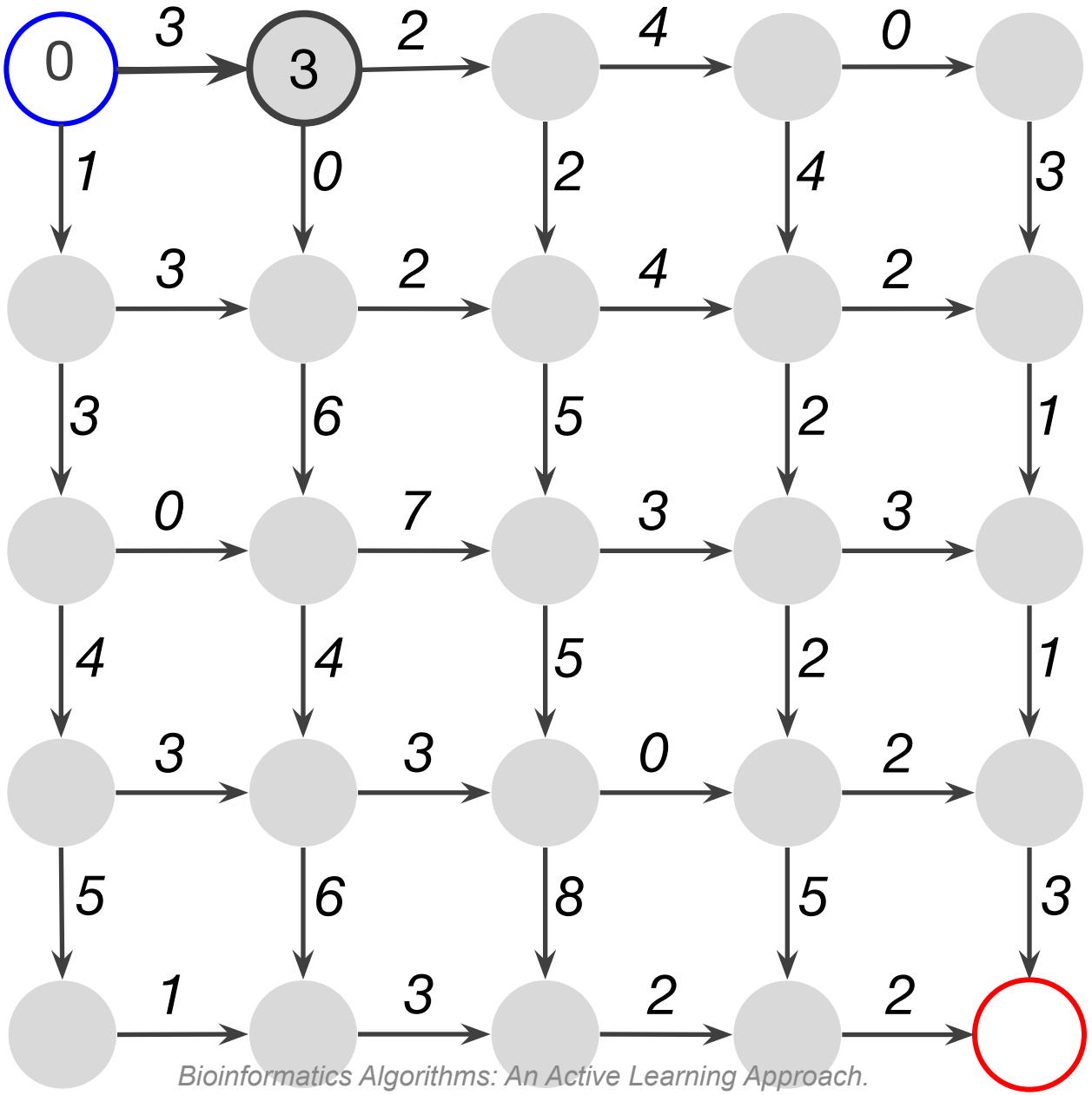


Manhattan Tourist Problem

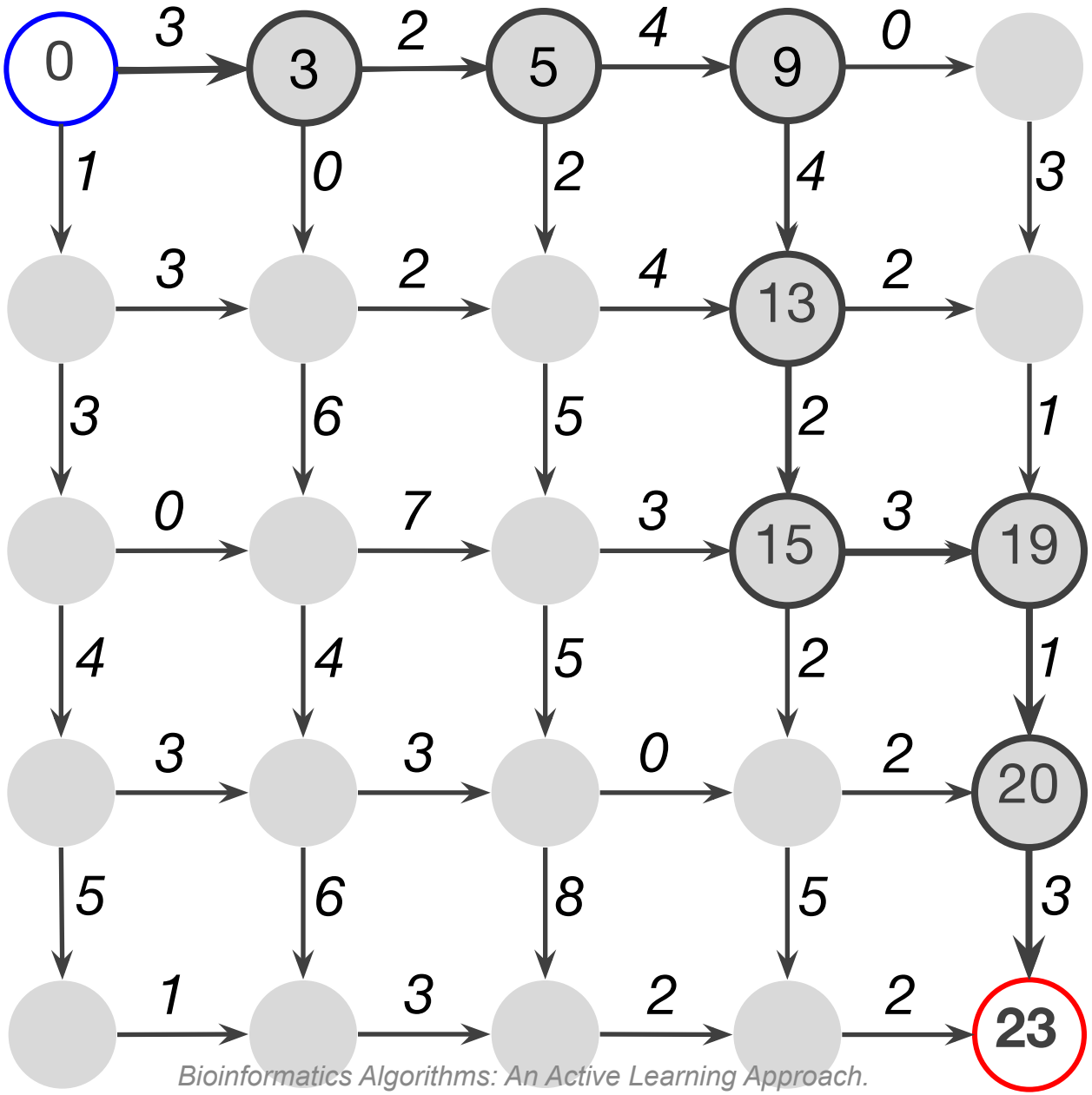
Manhattan Tourist Problem: Find a longest path in a rectangular city grid.

- **Input:** A weighted rectangular grid.
- **Output:** A longest path from the source to the sink in the grid.

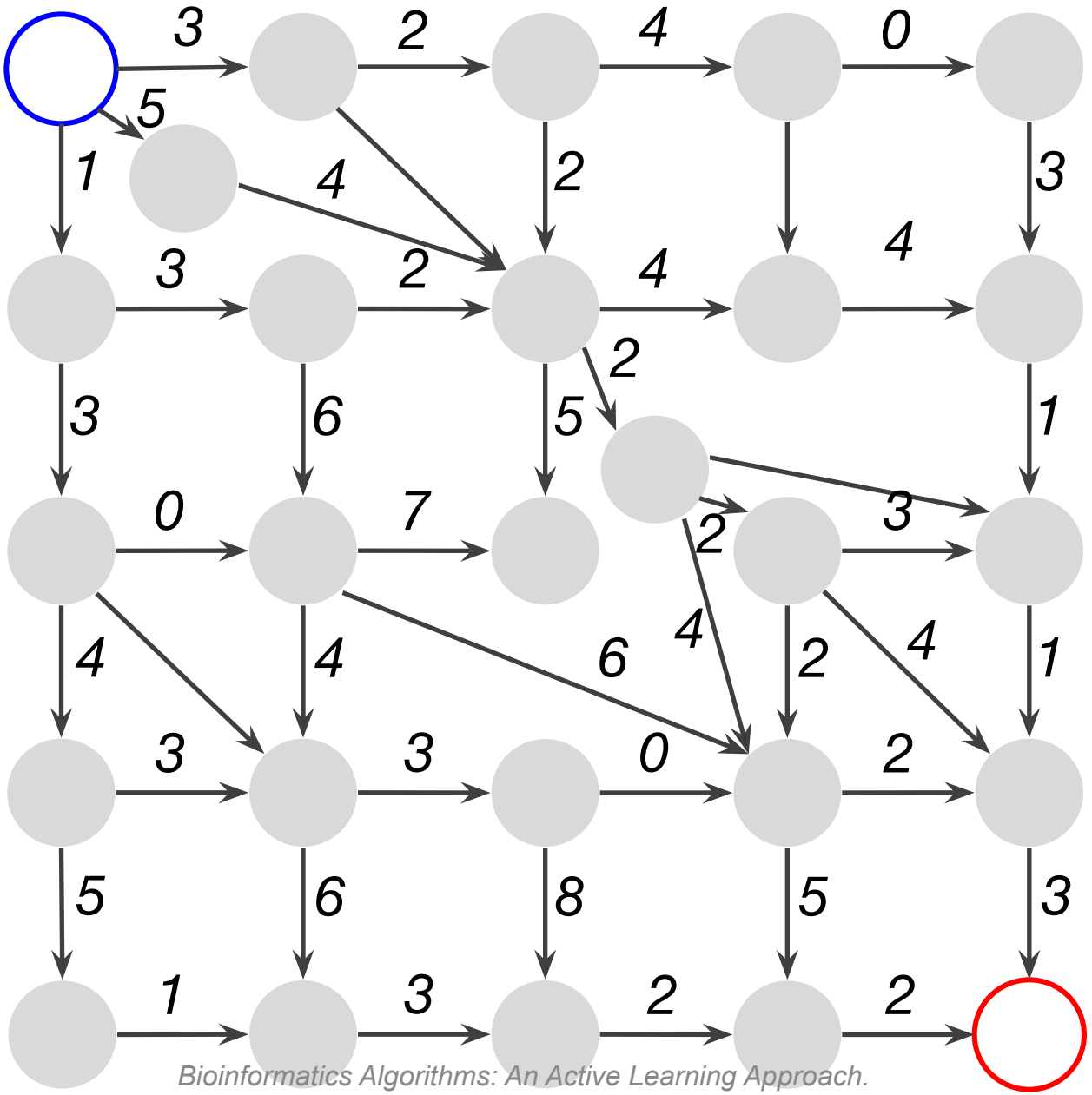
Greedy algorithm?



Greedy algorithm?



From a regular to an irregular grid

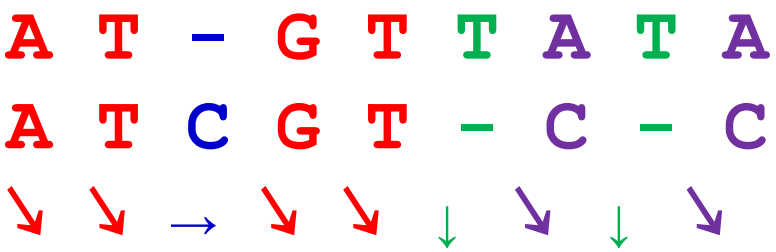


Search for Longest Paths in a Directed Graph

Longest Path in a Directed Graph Problem: Find a longest path between two nodes in an edge-weighted directed graph.

- **Input:** An edge-weighted directed graph with source and sink nodes.
- **Output:** A longest path from source to sink in the directed graph.

Do You See a Connection between the Manhattan Tourist and the Alignment Game?



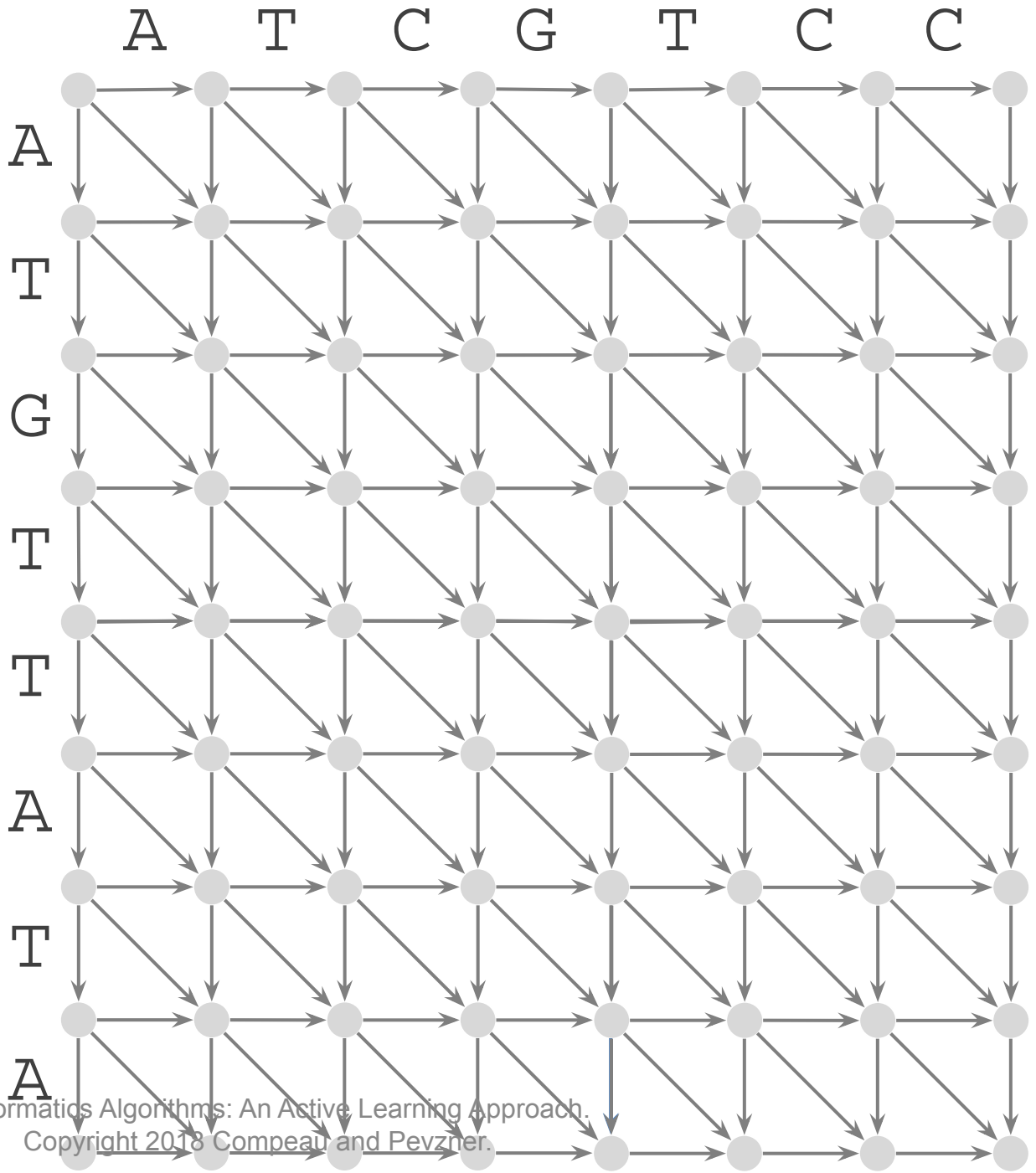
?

alignment → path

A T - G T T A T A

A T C G T - C - C

↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘



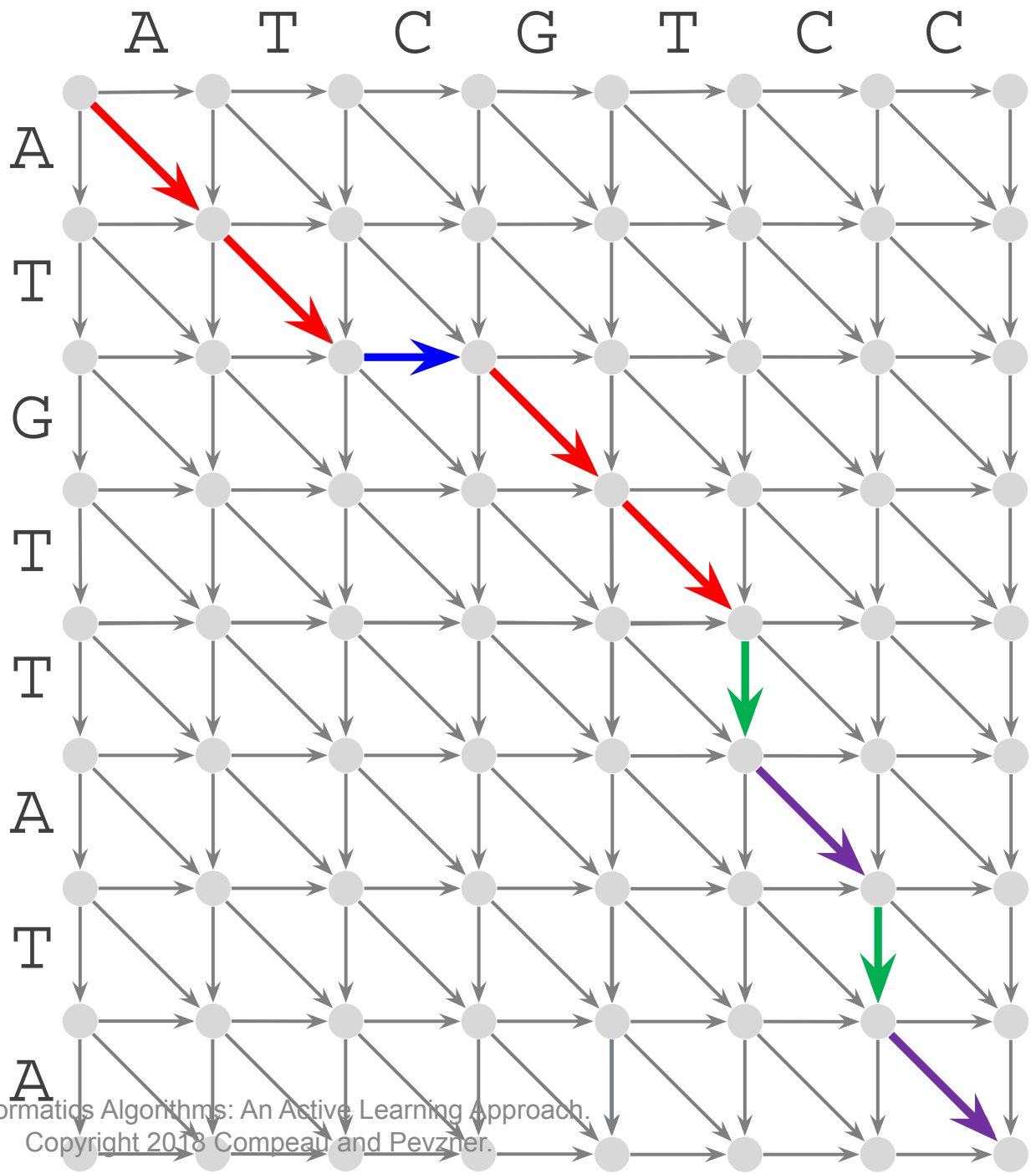
?

alignment → path

A T - G T T A T A

A T C G T - C - C

↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘



?

path ⟶ alignment

A T G T T - A T A

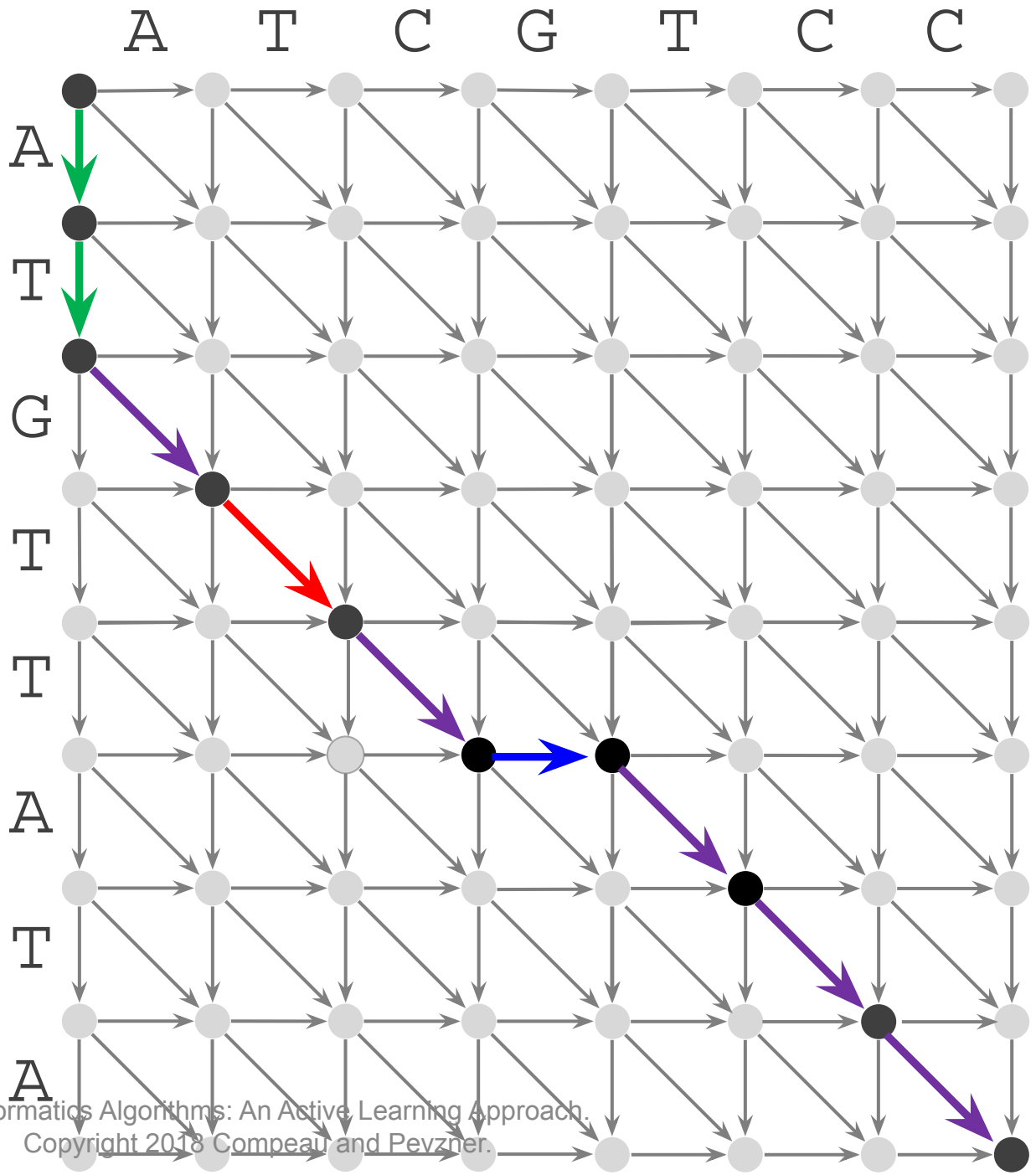
- - A T C G T C C

↓ ↓ ↘ ↘ ↘ → ↘ ↘ ↘

highest-scoring alignment

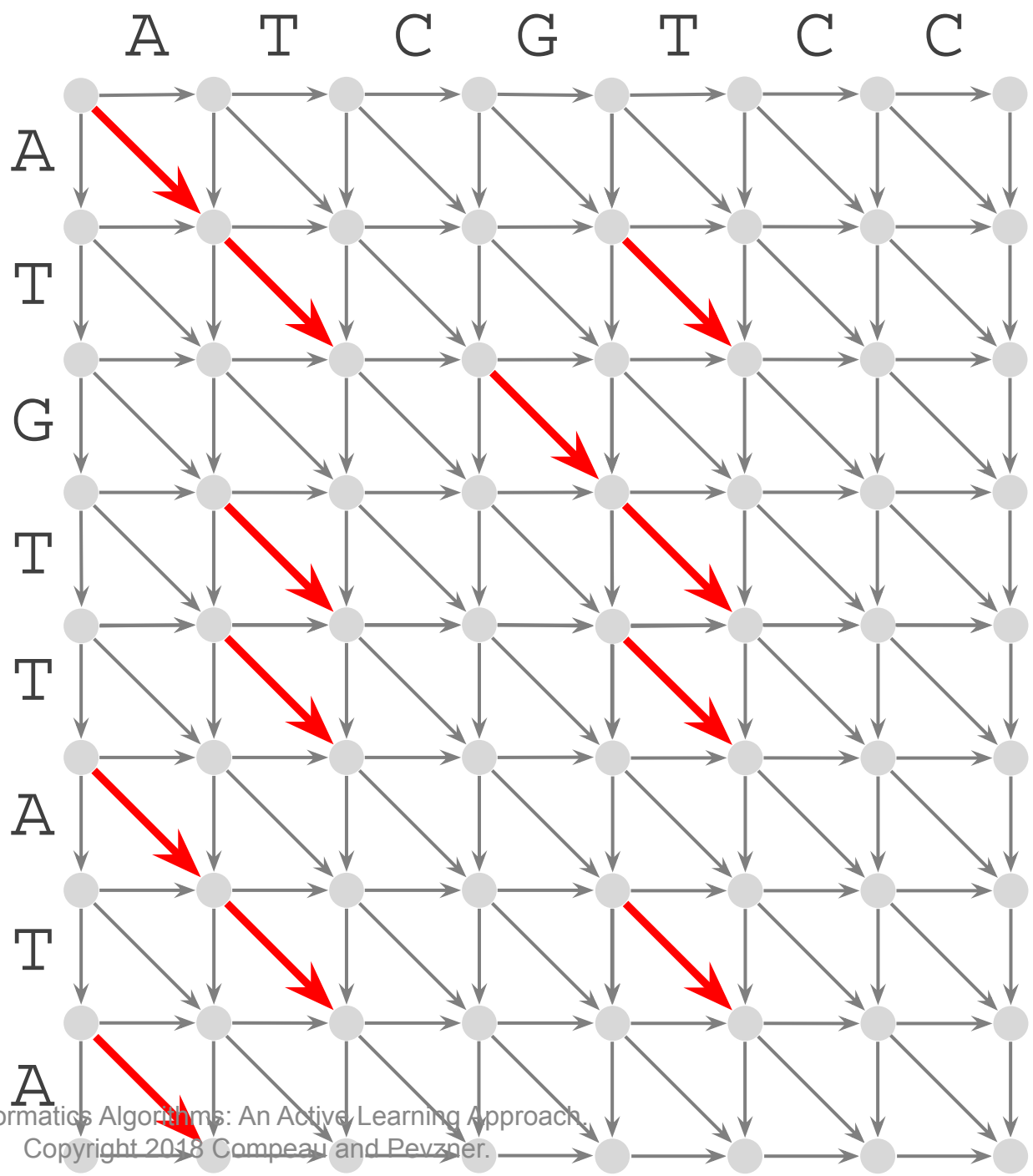
=

longest path in a properly built Manhattan



How to built a Manhattan for the Alignment Game and the Longest Common Subsequence Problem?

Diagonal red edges correspond to matching symbols and have scores 1



How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- **The Change Problem**
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

The Change Problem

Change Problem: Find the minimum number of coins needed to make change.

- **Input:** An integer *money* and an array of positive integers ($coin_1, \ldots, coin_d$).
- **Output:** The minimum number of coins with denominations ($coin_1, \ldots, coin_d$) that changes *money*.

Changing Money in a Greedy Way

```
GreedyChange(money)
  change ← empty collection of coins
  while money > 0
    coin ← largest denomination that does not exceed money
    add coin to change
    money ← money – coin
  return change
```

Changing Money in Tanzania

40 cents = 25+10+5

Greedy

Changing Money in Tanzania: **GreedyChange** Fails

40 cents = 25+10+5 = 20+20

Greedy is not **Optimal**

Recursive Change

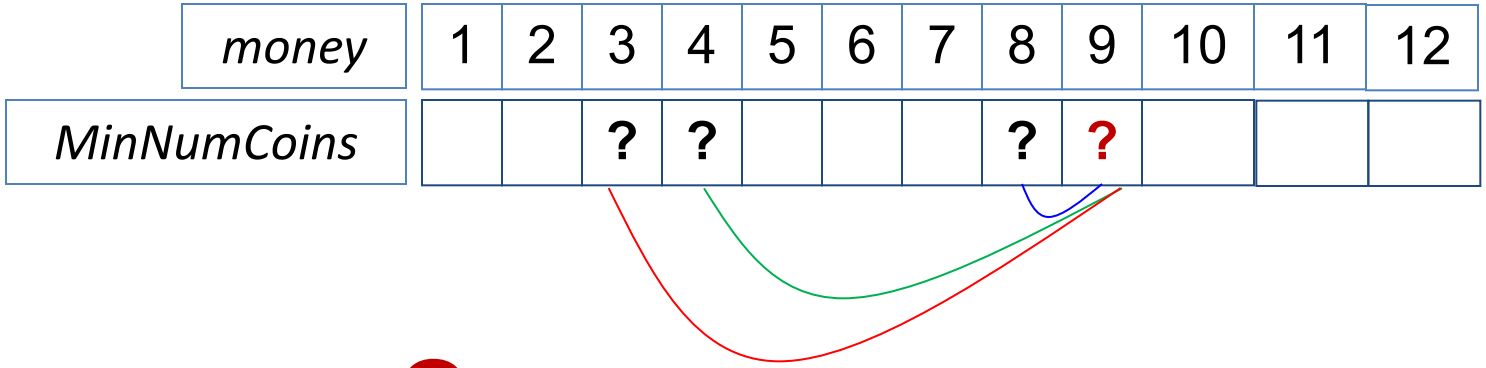
Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins									?			

MinNumCoins(9) = ?

Recursive Change

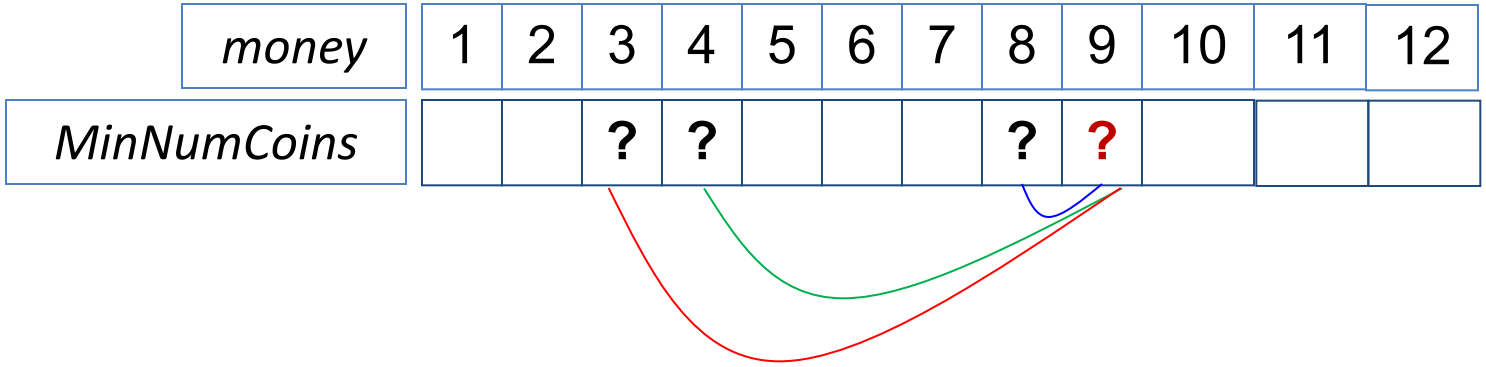
Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?



MinNumCoins(9) = ?

Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?



$$MinNumCoins(9) = \min \begin{cases} MinNumCoins(9-6)+1 = MinNumCoins(3)+1 \\ MinNumCoins(9-5)+1 = MinNumCoins(4)+1 \\ MinNumCoins(9-1)+1 = MinNumCoins(8)+1 \end{cases}$$

Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins			?	?				?	?			

MinNumCoins(3) =
MinNumCoins(4) =
MinNumCoins(8) =

?

Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins		?	?	?			?	?	?			

MinNumCoins(3) =
MinNumCoins(4) =
MinNumCoins(8) =

?

Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins		?	?	?			?	?	?			

$$\textit{MinNumCoins}(money) = \min \begin{cases} \textit{MinNumCoins}(money-6) + 1 \\ \textit{MinNumCoins}(money-5) + 1 \\ \textit{MinNumCoins}(money-1) + 1 \end{cases}$$

Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins		?	?	?			?	?	?			

$$\textit{MinNumCoins}(money) = \begin{cases} \\ \min_i \{ \textit{MinNumCoins}(money - coin_i) \} + 1 \\ \\ \end{cases}$$

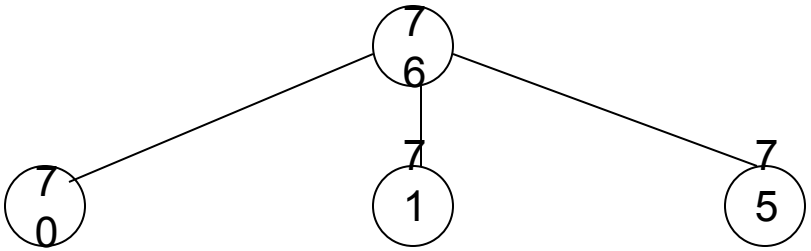
$$\textit{MinNumCoins}(money - coin_d) + 1$$

RecursiveChange

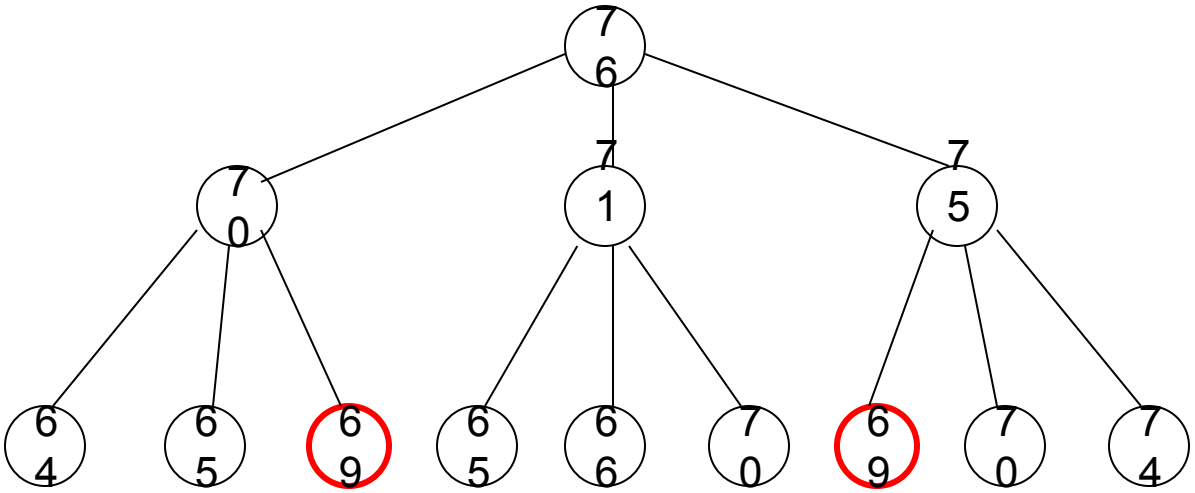
```
RecursiveChange(money, coins)
    if money = 0
        return 0
    MinNumCoins ← infinity
    for i ← 1 to |coins|
        if money ≥ coin_i
            NumCoins ←
RecursiveChange(money-coin_i, coins)
            if numCoins + 1 < MinNumCoins
                MinNumCoins ← numCoins + 1
    return MinNumCoins
```

How Fast is the RecursiveChange?

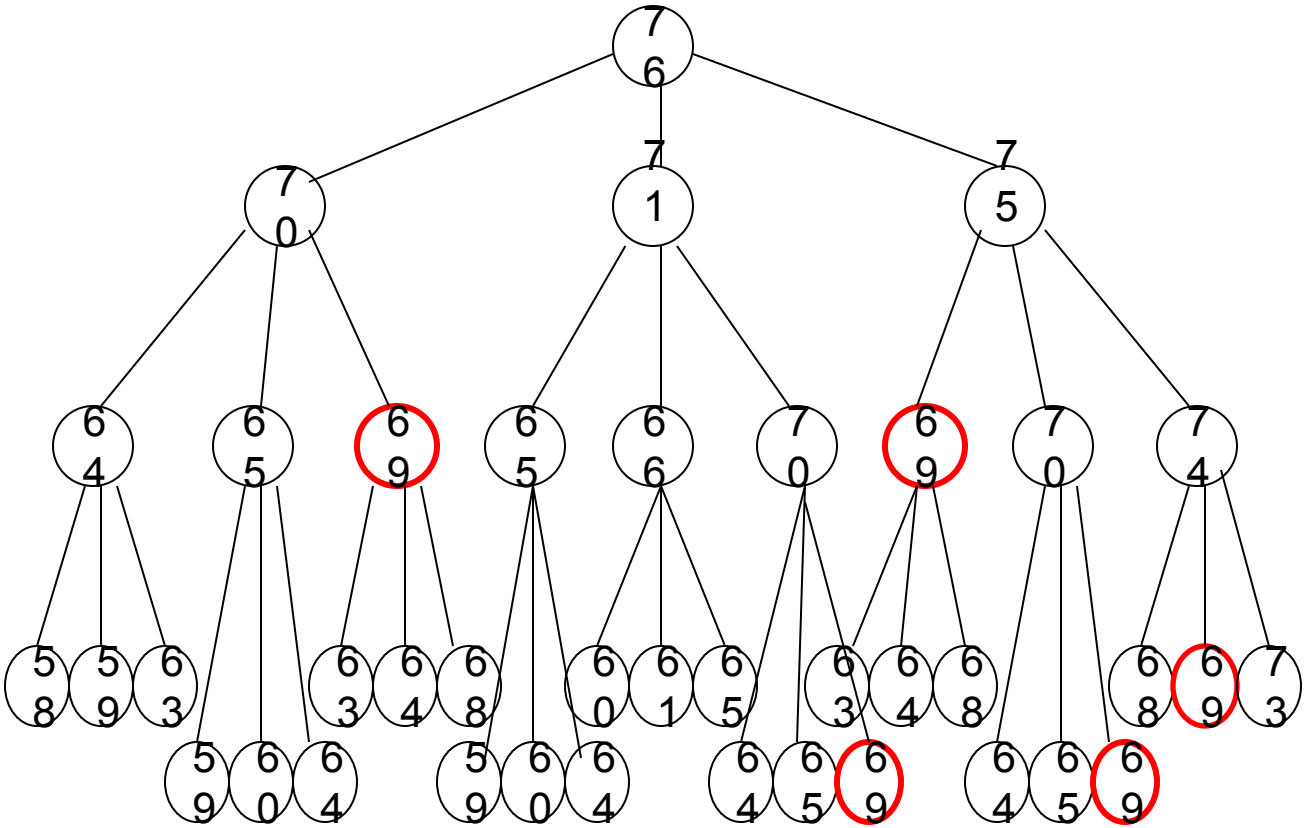
The Recursive Tree



The Recursive Tree



The Recursive Tree



the optimal coin combination for 69 cents is computed **6** times!

the optimal coin combination for 30 cents is computed **trillions** of times!

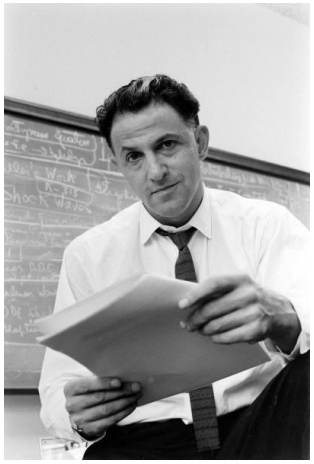
Changing Money by Dynamic Programming

Hint. Wouldn't it be nice to know all the values of

$$MinNumCoins(money - coin_i)$$

by the time we need to compute

$$MinNumCoins(money)?$$



Richard Bellman

Instead of the time-consuming calls:

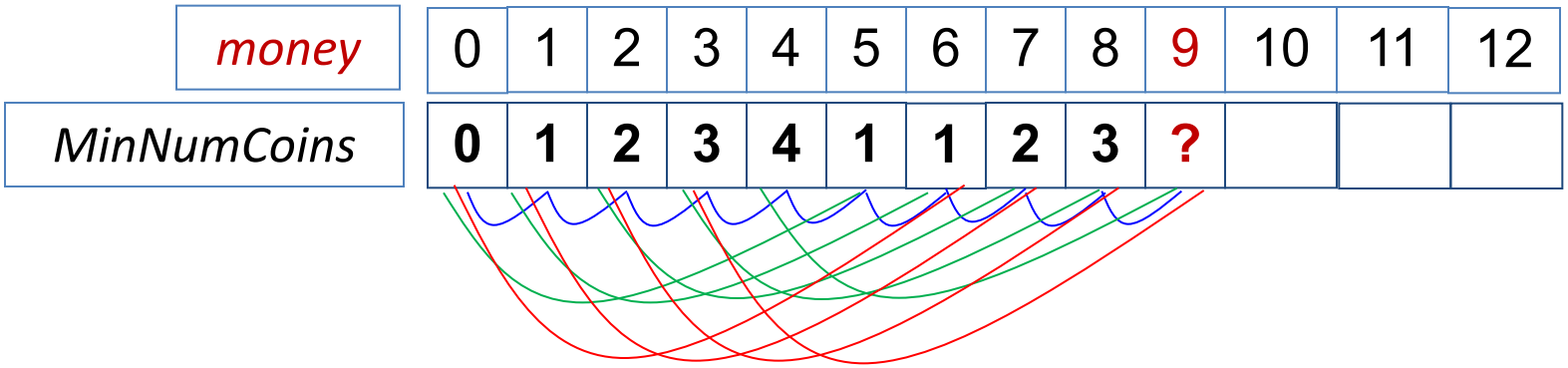
RecursiveChange$(money\text{-}coin_i,$ ***Coins***$, d)$

we would simply look up the values of

$$MinNumCoins(money - coin_i)$$

Changing Money by Dynamic Programming

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?



DPChange

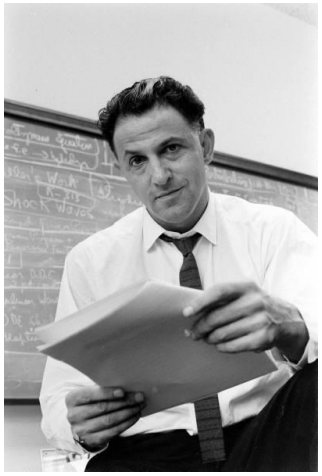
DPChange(*money, **coins***)
  MinNumCoins(0) □ 0
  for *m* □ 1 to *money*
    MinNumCoins(*m*) □ infinity
    for *i* □ 1 to |***coins***|
      if $m \geq coin_i$
        if $MinNumCoins(m - coin_i) + 1 < MinNumCoins(m)$
          $MinNumCoins(m) \square MinNumCoins(m - coin_i) + 1$
  return *MinNumCoins*(*money*)

“Programming” in “Dynamic Programming” Has Nothing to Do with Programming!

Richard Bellman developed this idea in 1950s working on an Air Force project.

At that time, his approach seemed completely impractical.

He wanted to hide that he is really doing math from the Secretary of Defense.



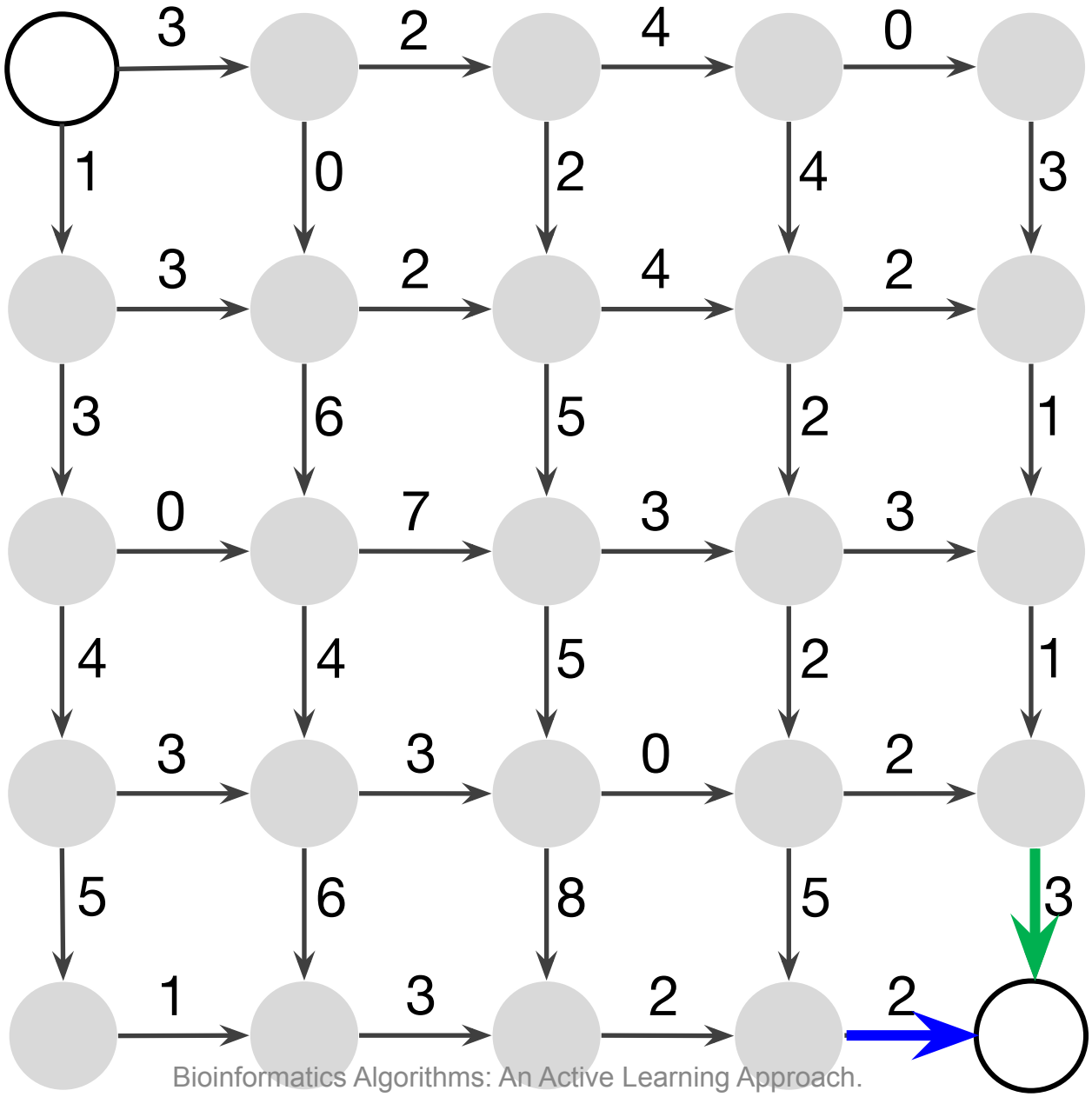
Richard Bellman

“…What name could I choose? I was interested in planning but planning, is not a good word for various reasons. I decided therefore to use the word, “programming” and I wanted to get across the idea that this was dynamic. **It was something not even a Congressman could object to. So I used it as an umbrella for my activities.”**

How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- **Dynamic Programming and Backtracking Pointers**
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

There are only 2 ways to arrive to the sink: by moving **South** ↓ or by moving **East** →

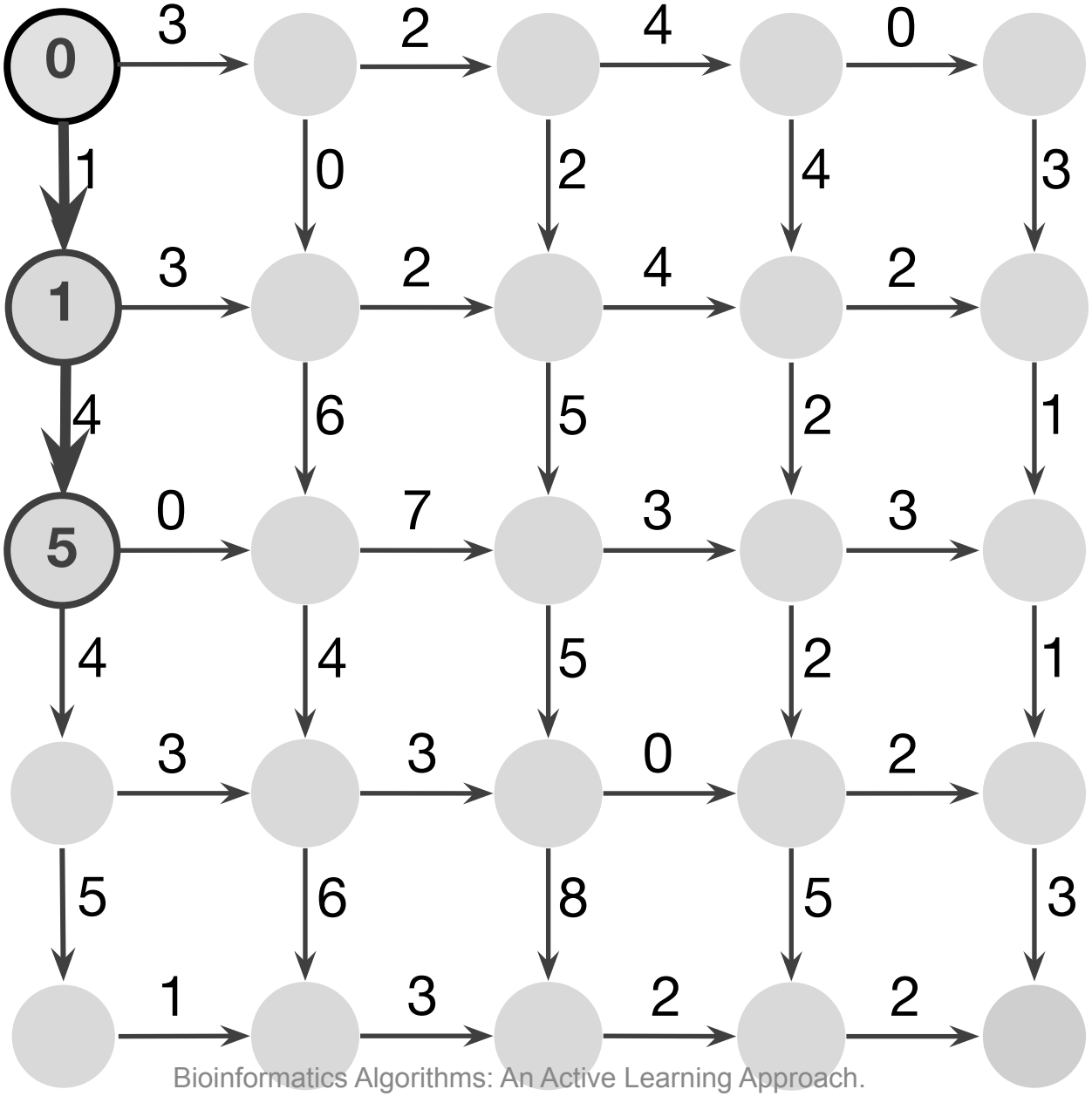


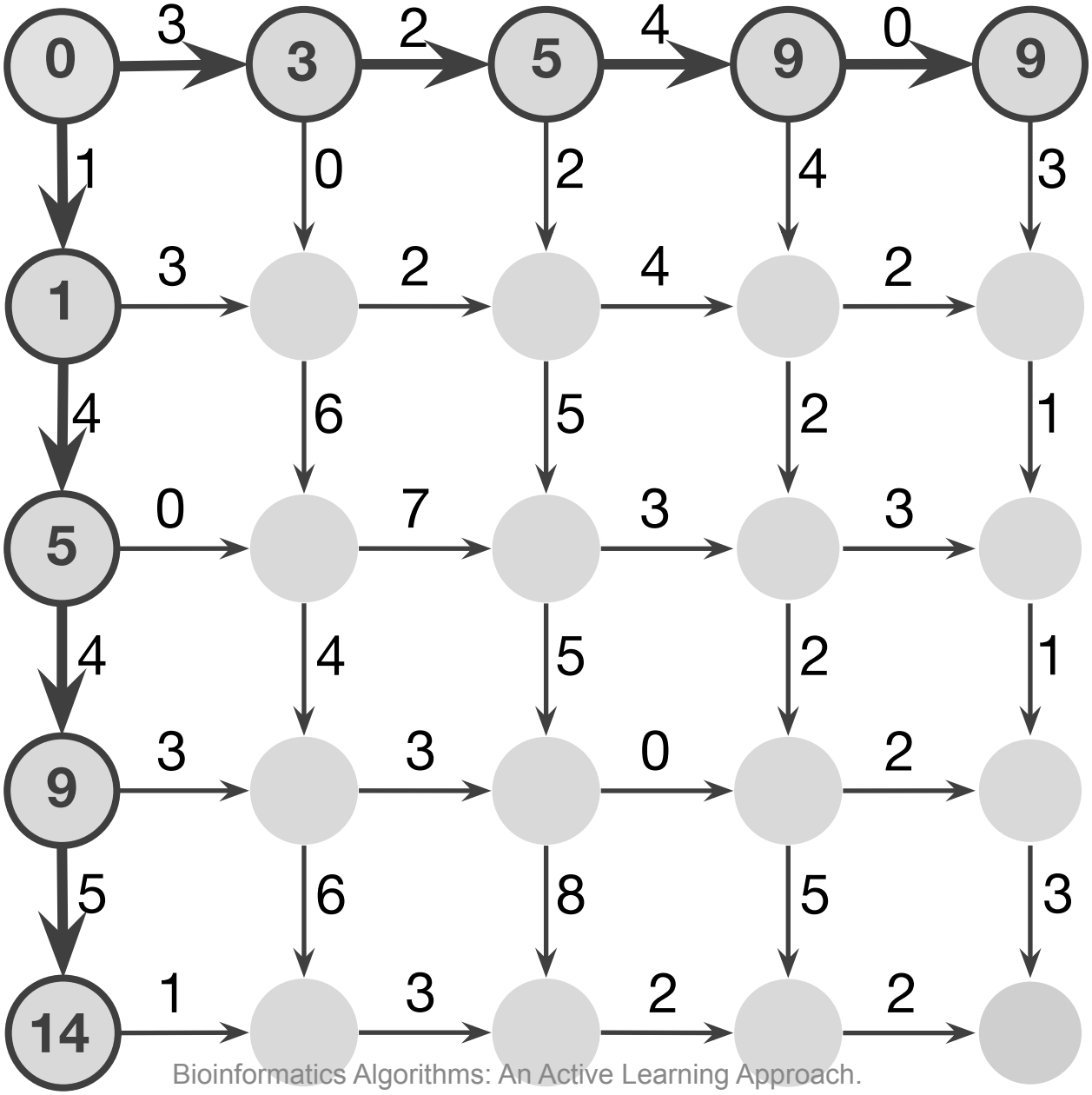
South or East?

South or East?

SouthOrEast(*n,m*)

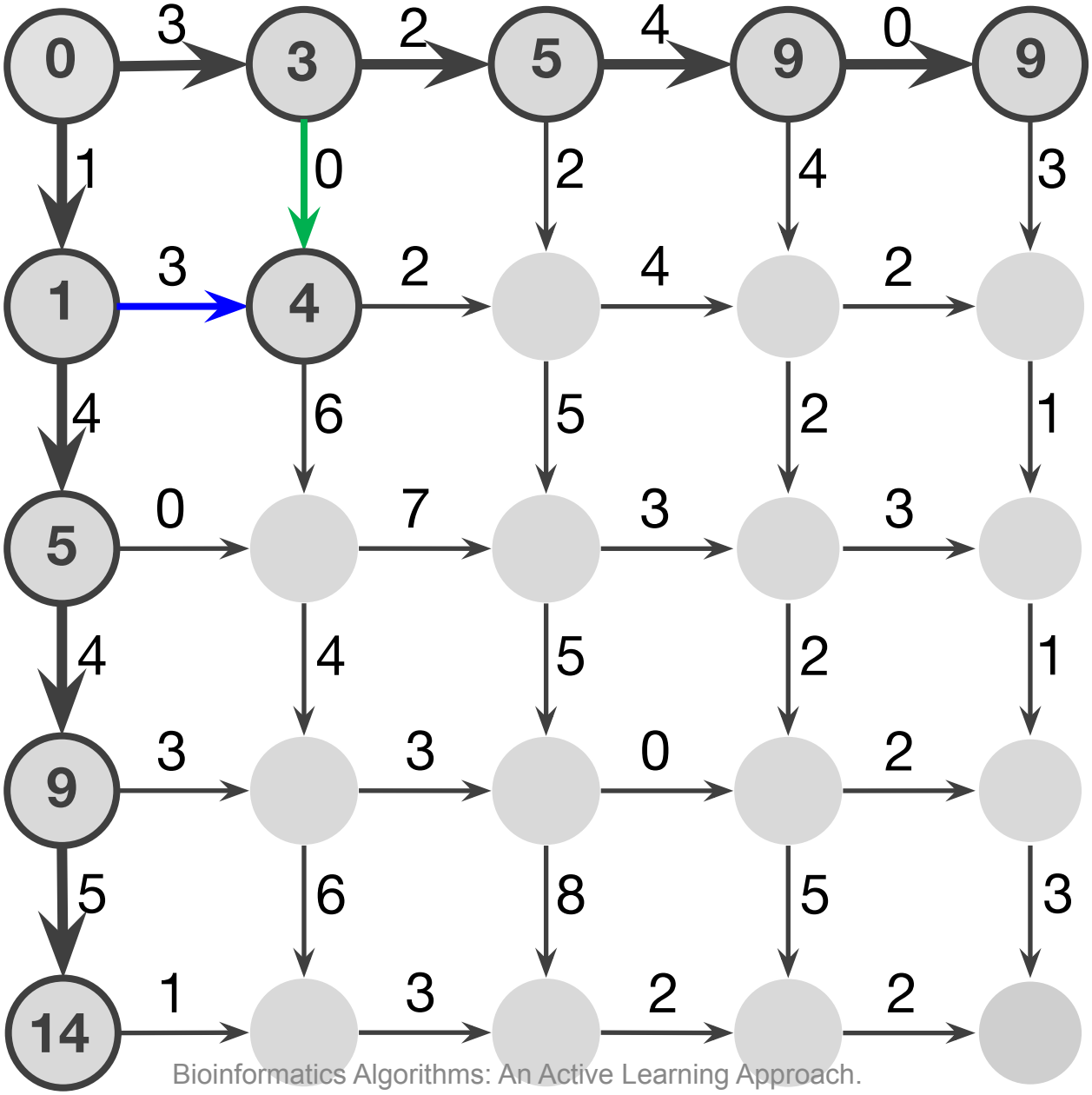
```
if n=0  and m=0
    return  0
if n>0  and m>0
    x ← SouthOrEast(n-1,m)+weight of edge "↓"into (n,m)
    y ← SouthOrEast(n,m-1)+ weight of edge "→"into (n,m)
    return max{x,y}
return -infinity
```



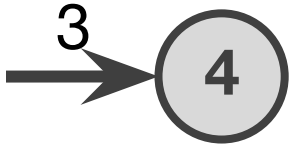


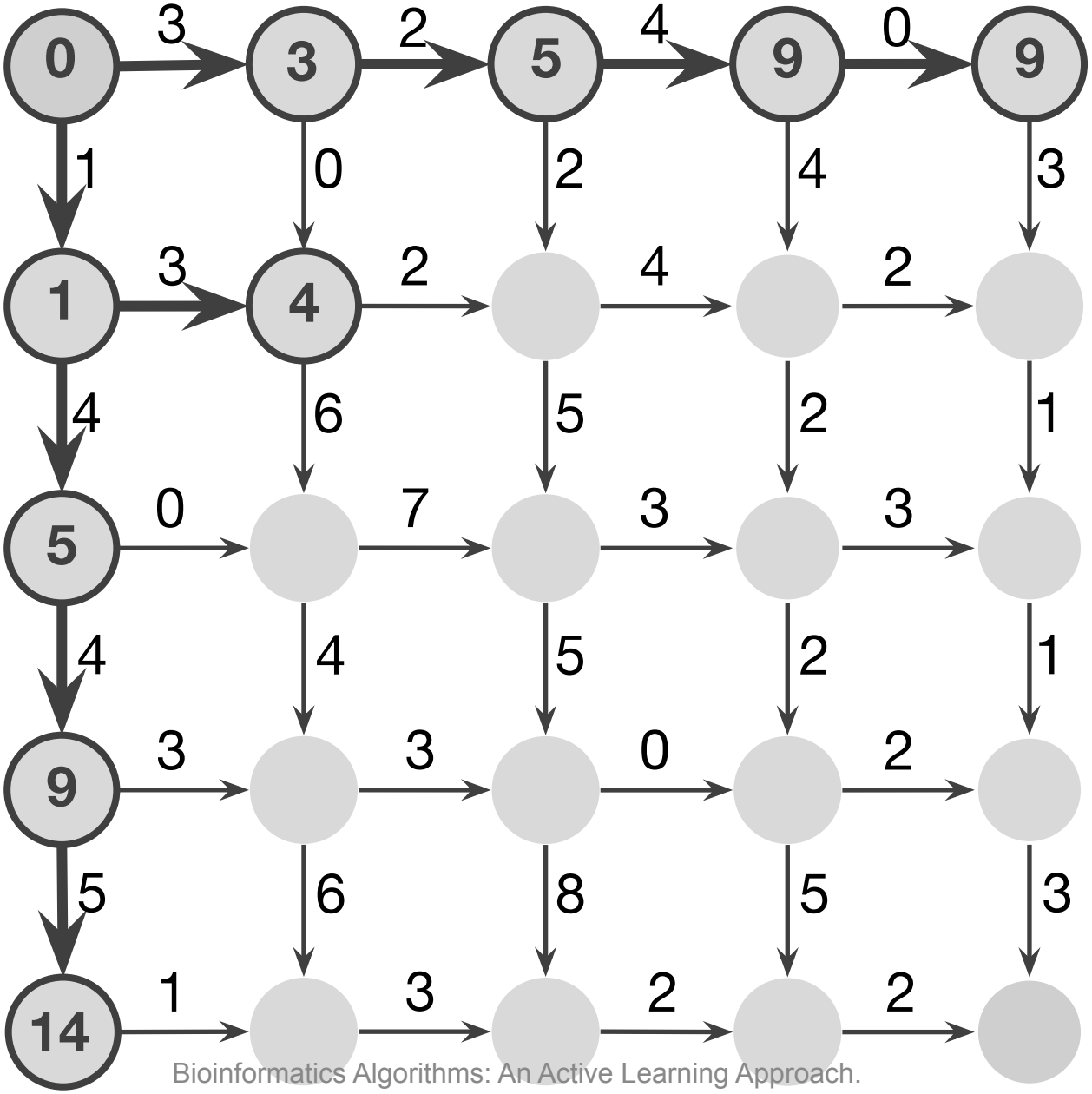
South
or
East?

1+3 > 3+0



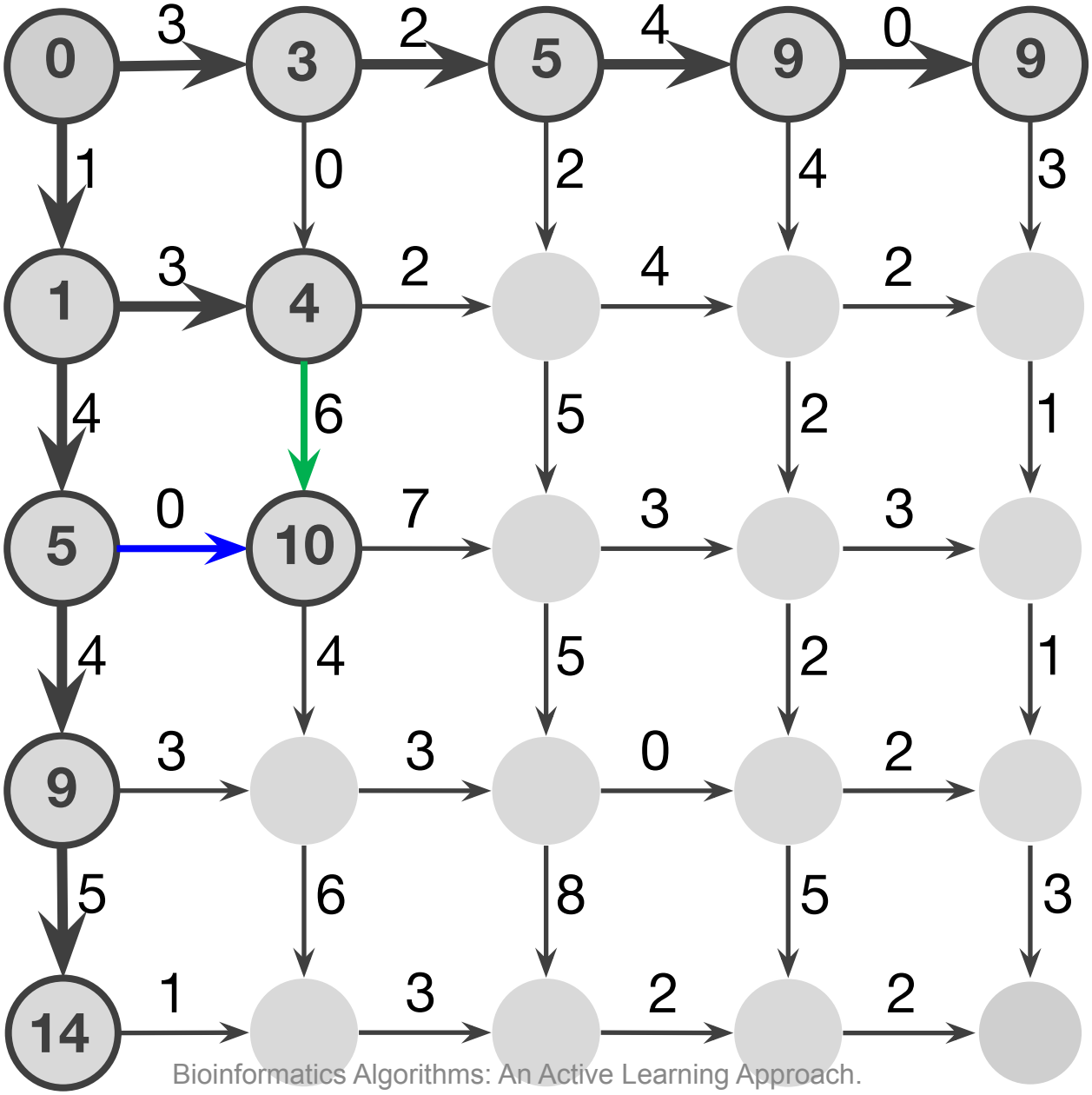
We arrived to (1,1) by the **bold** edge:



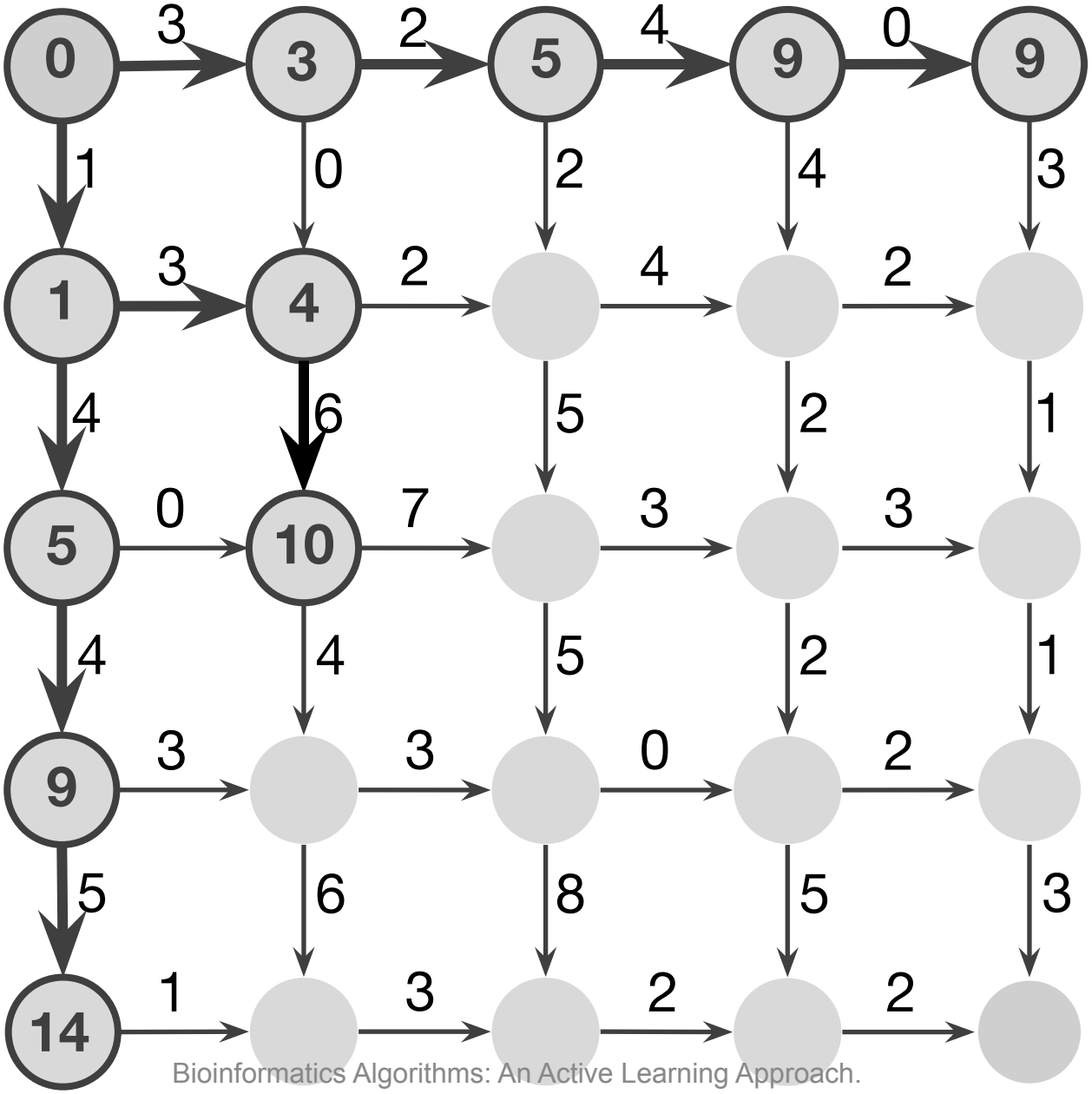


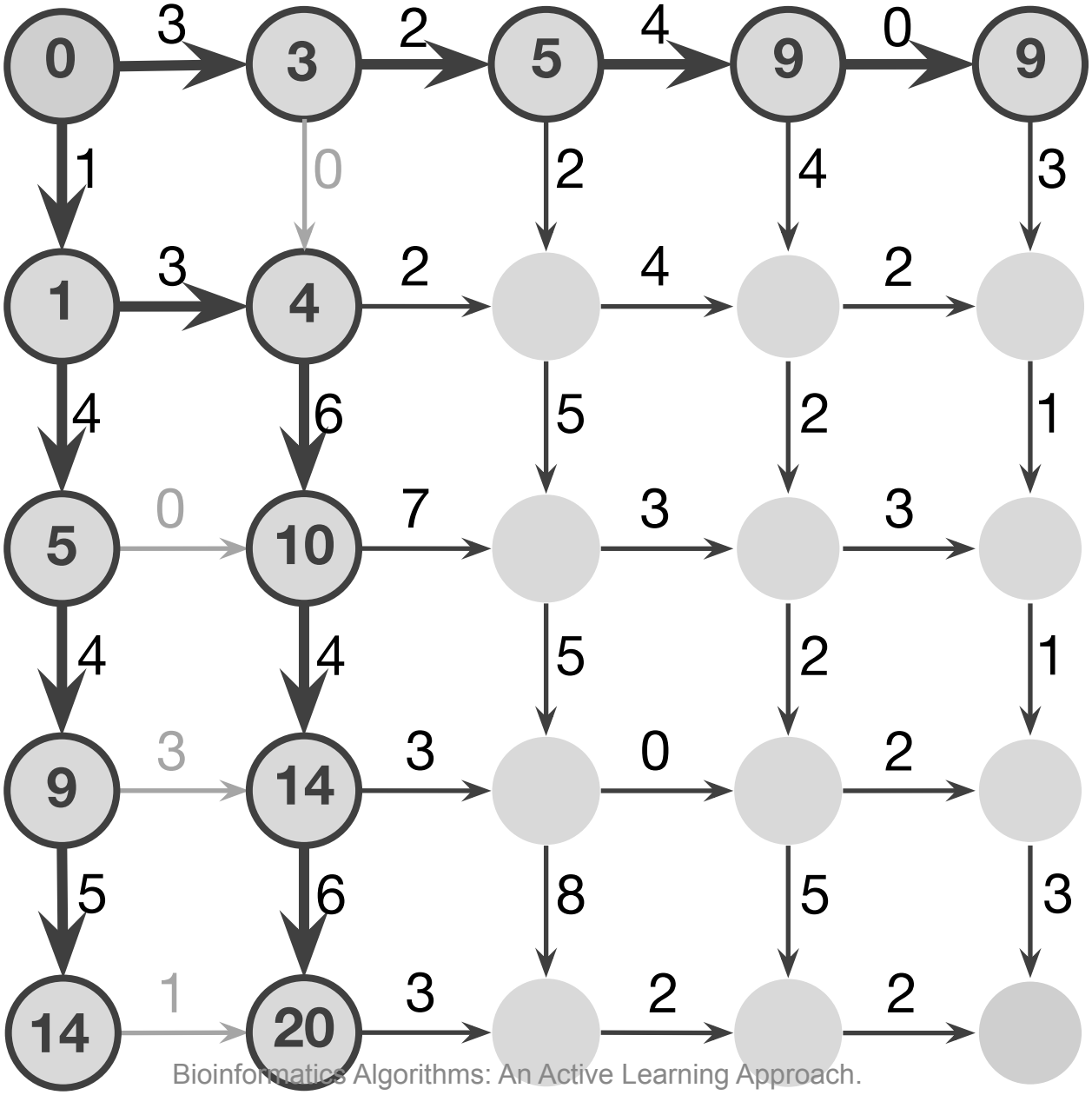
South
or
East?

5+0 < 4+6



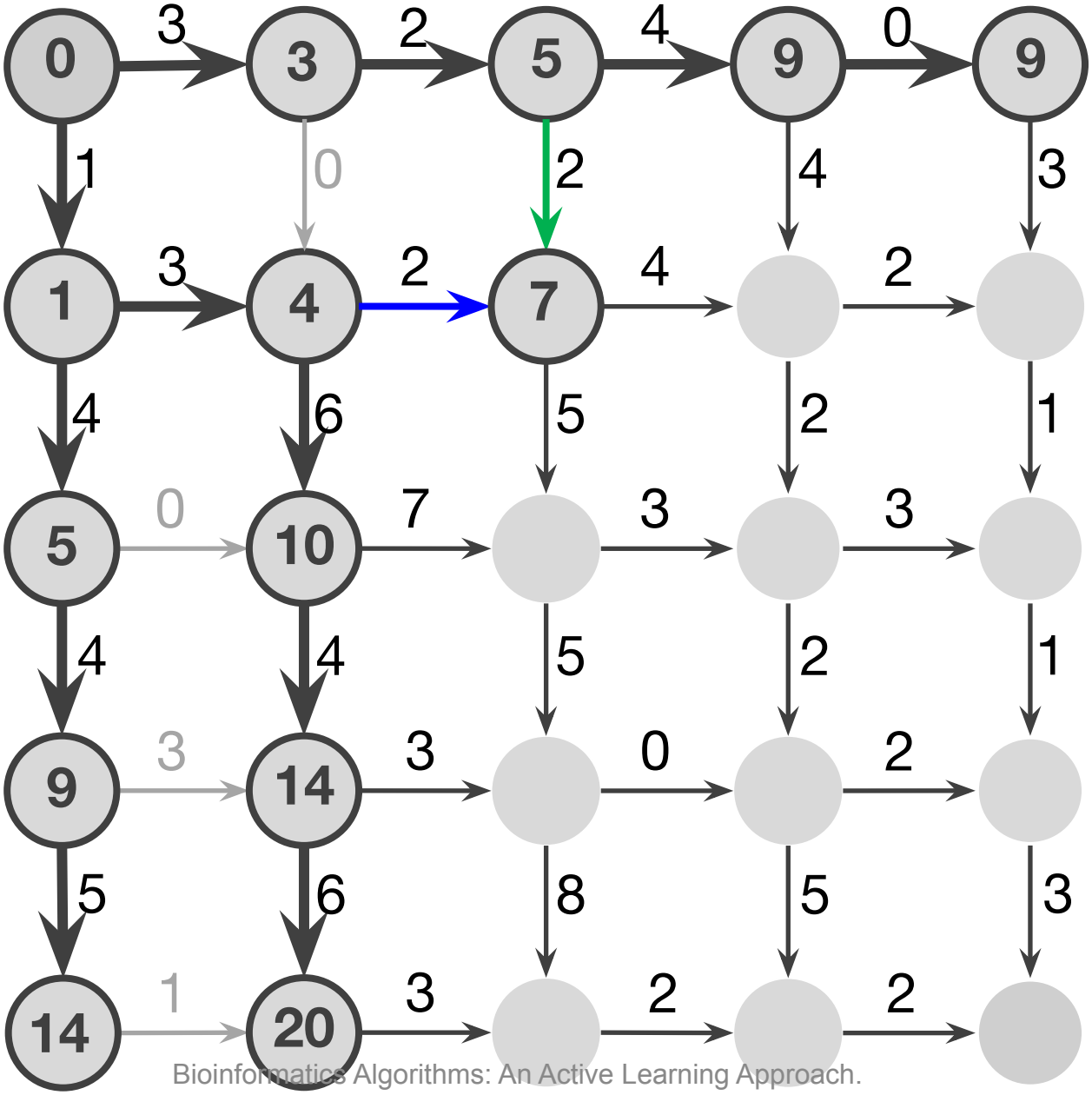
We arrived to (2,1) by the **bold** edge:





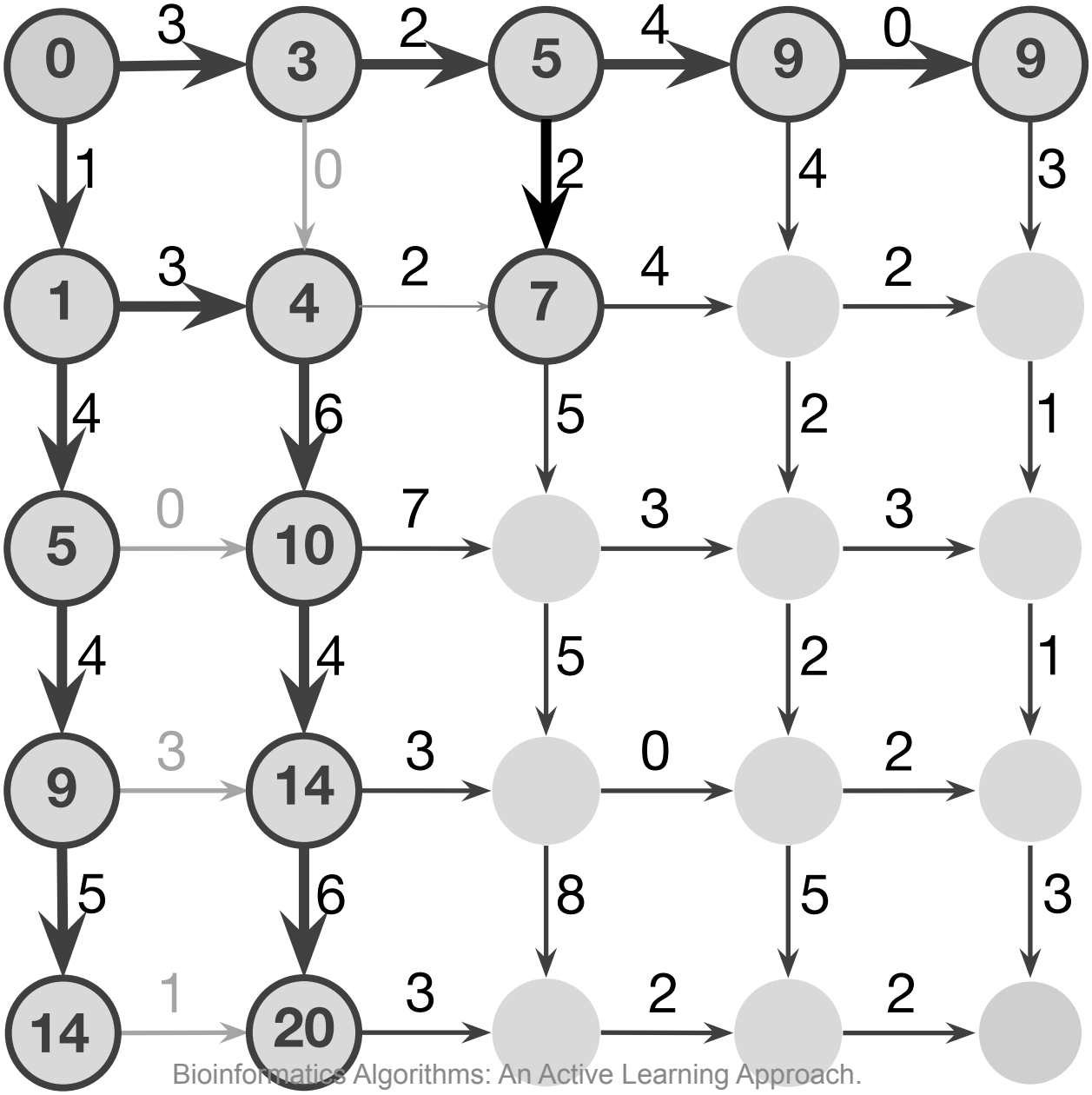
South
or
East?

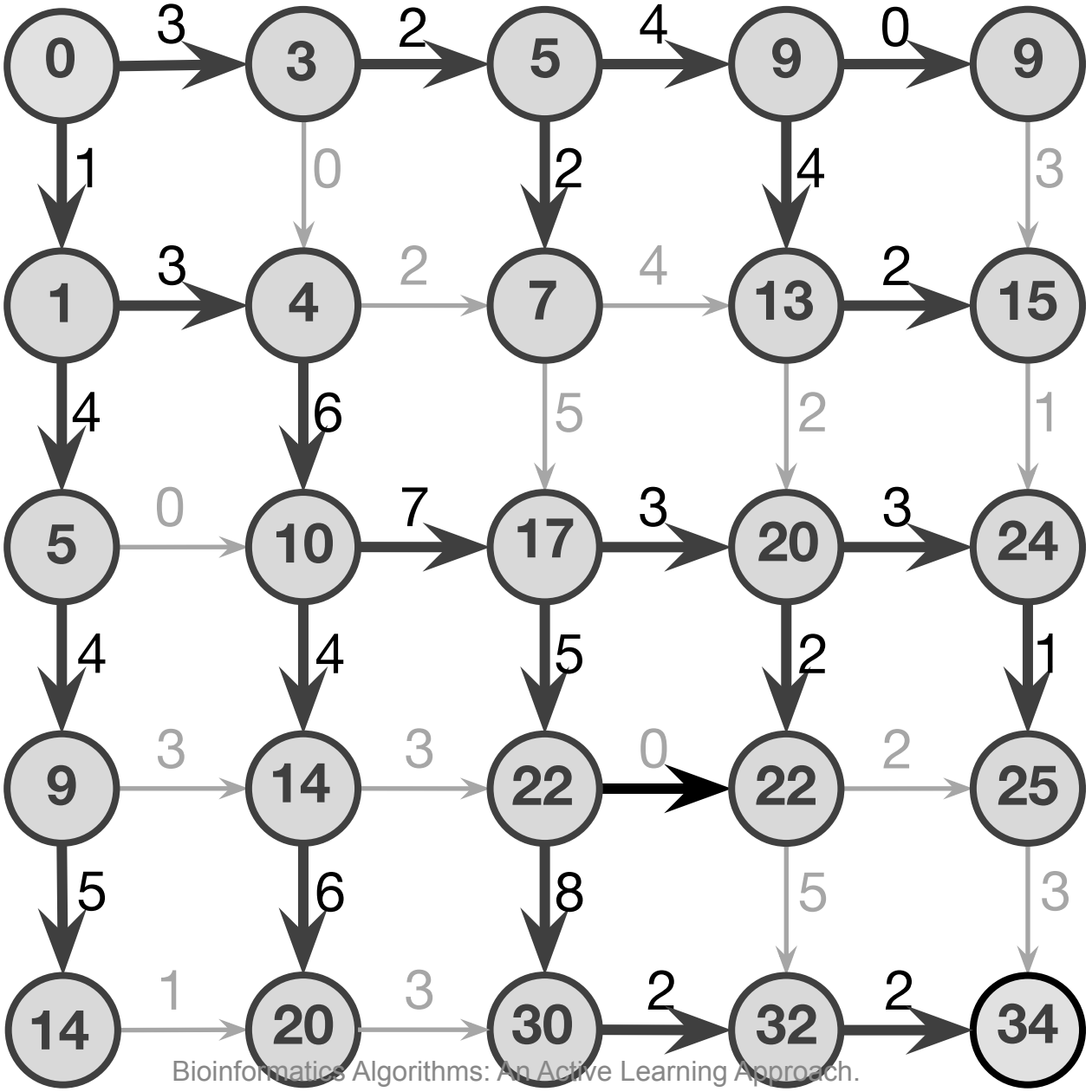
5+2 > 4+2



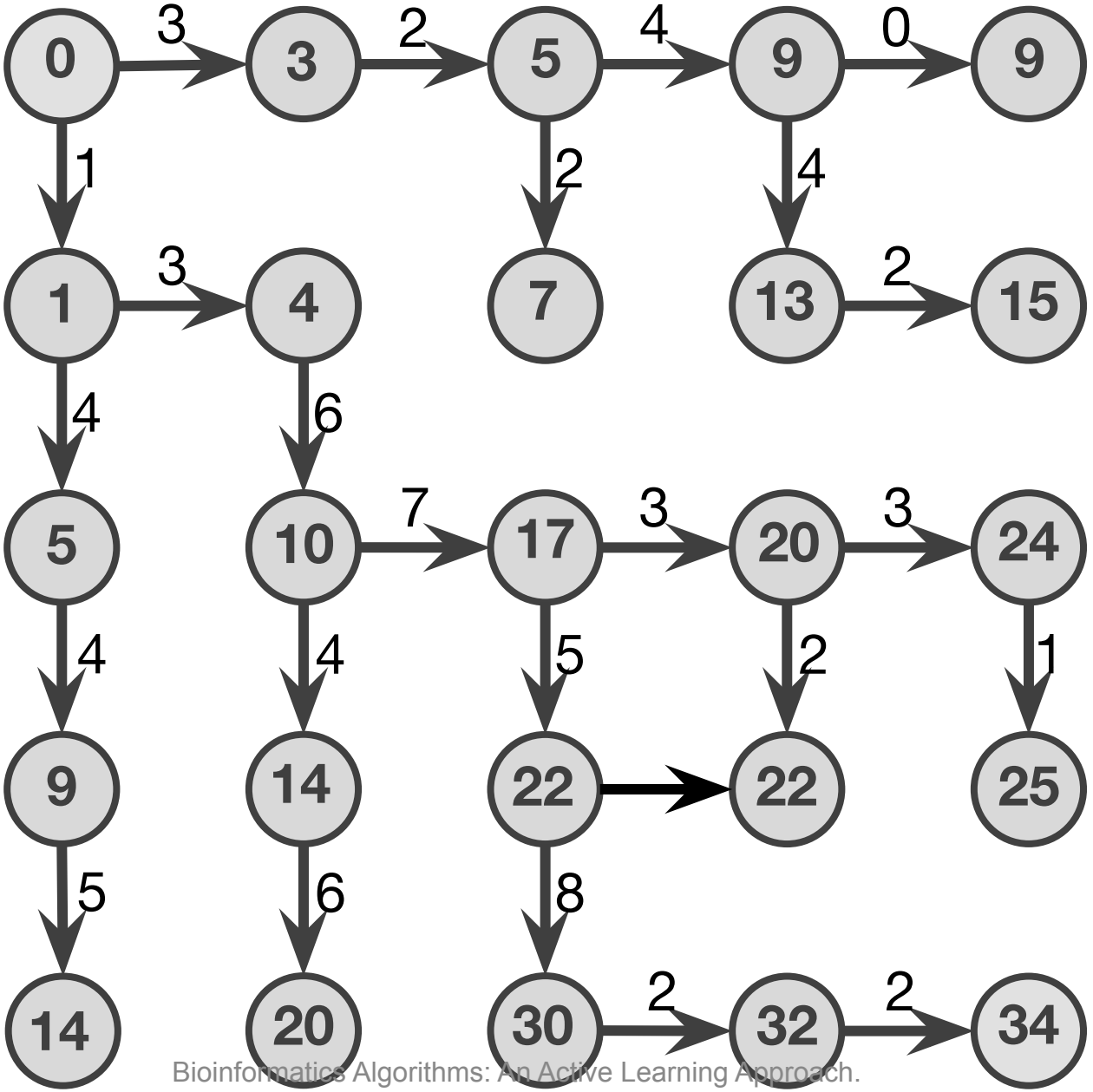
South
or
East?

5+2 > 4+2





Backtracking pointers: the best way to get to each node



Dynamic Programming Recurrence

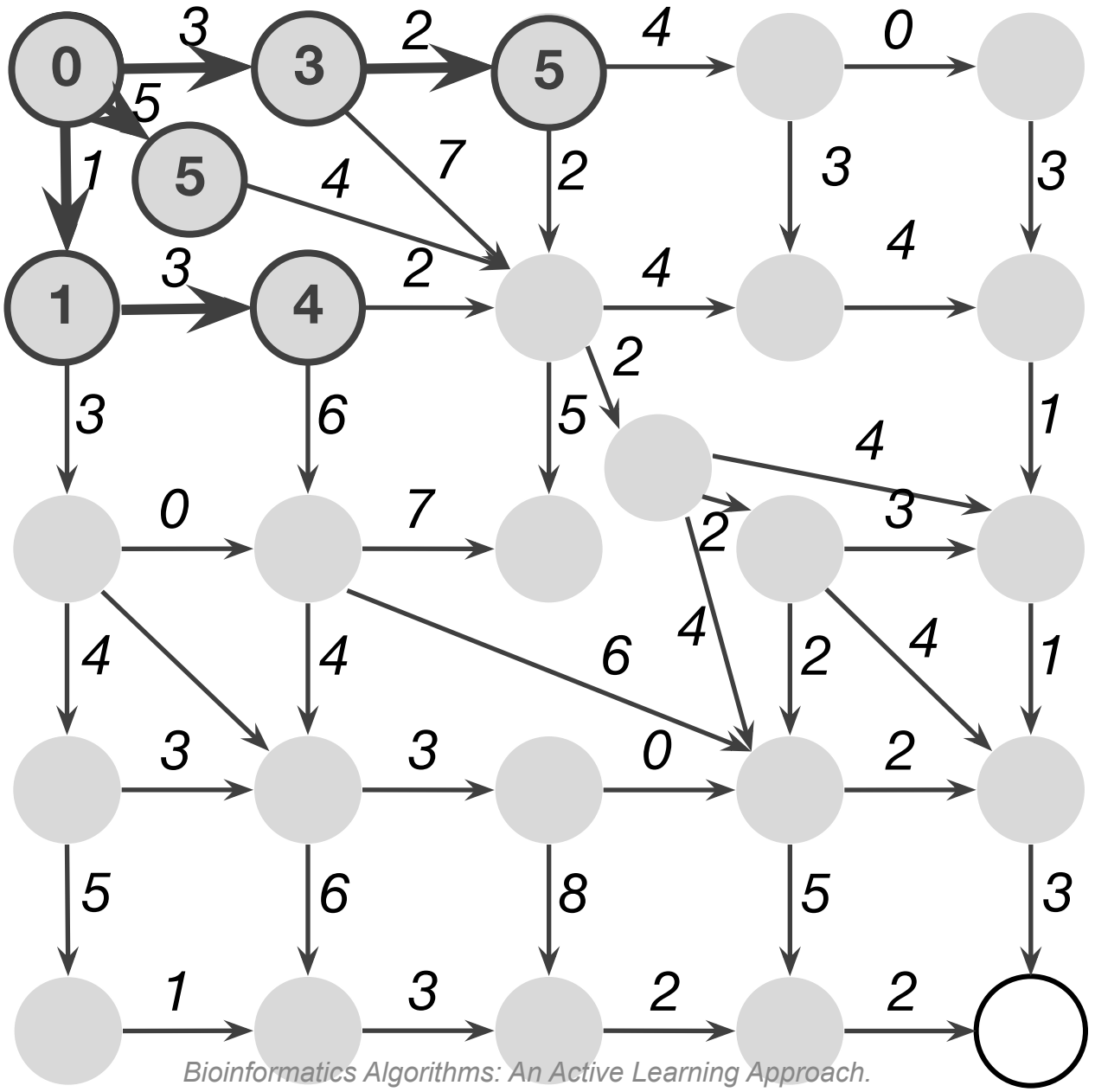
$s_{i,j}$: the length of a longest path from (0,0) to (i,j)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \end{cases}$$

How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- **From Manhattan to the Alignment Graph**
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

How does the recurrence change for this graph?



$s_a = \max_{\text{all predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$

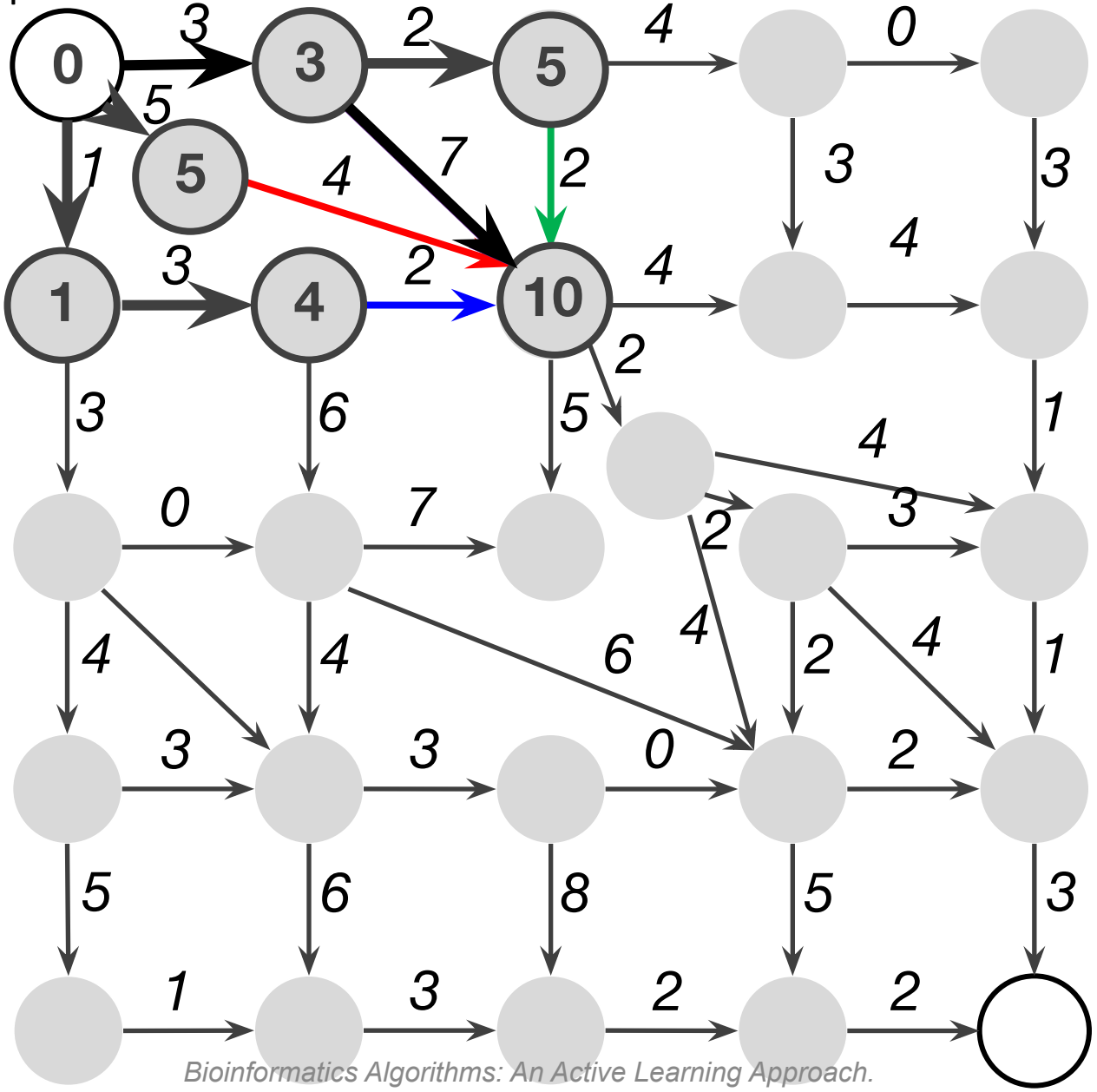
4 choices:

5 + 2

3 + 7

5 + 4

4 + 2



$s_a = \max_{\text{all predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$

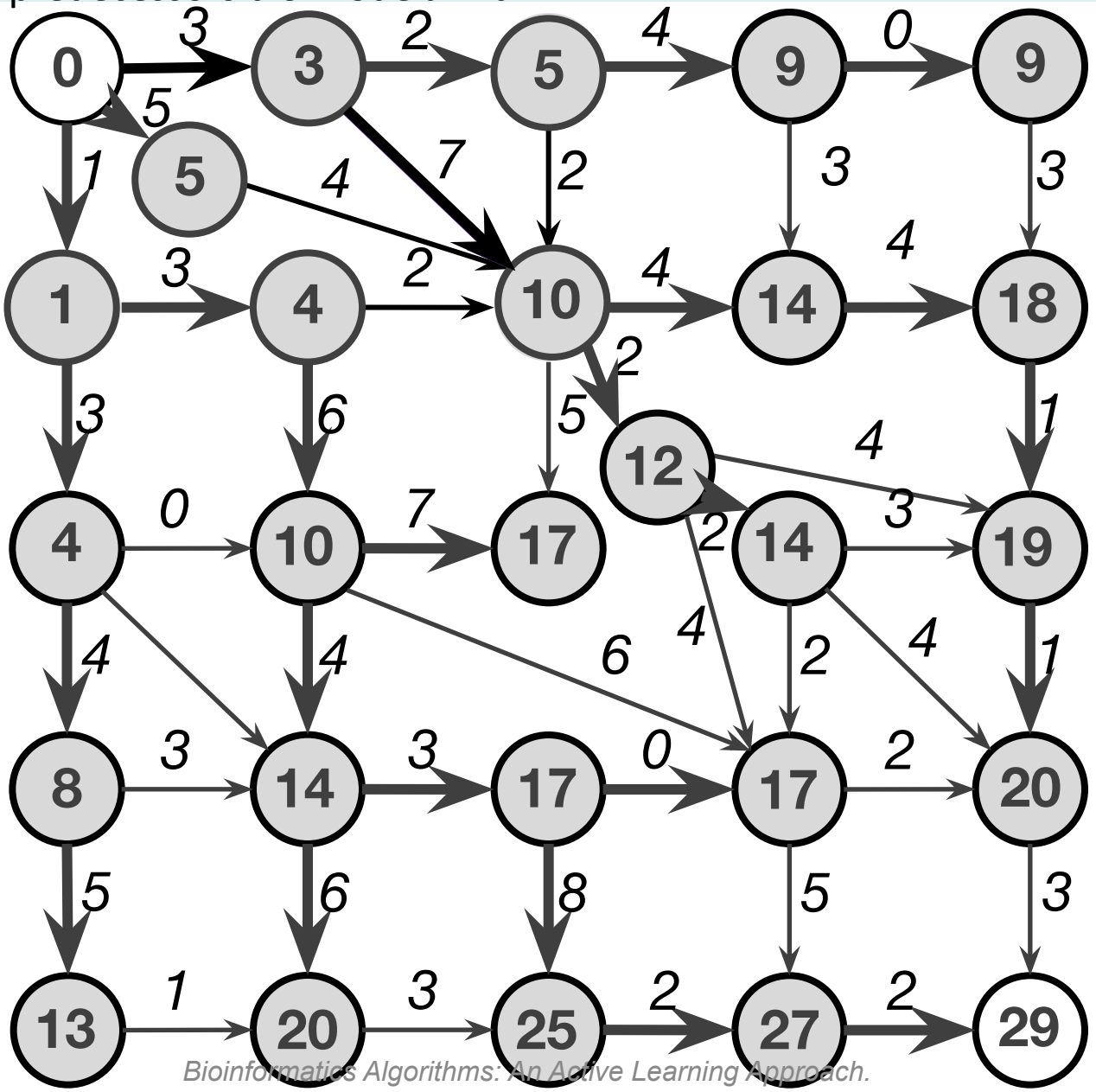
4 choices:

5 + 2

3 + 7

5 + 4

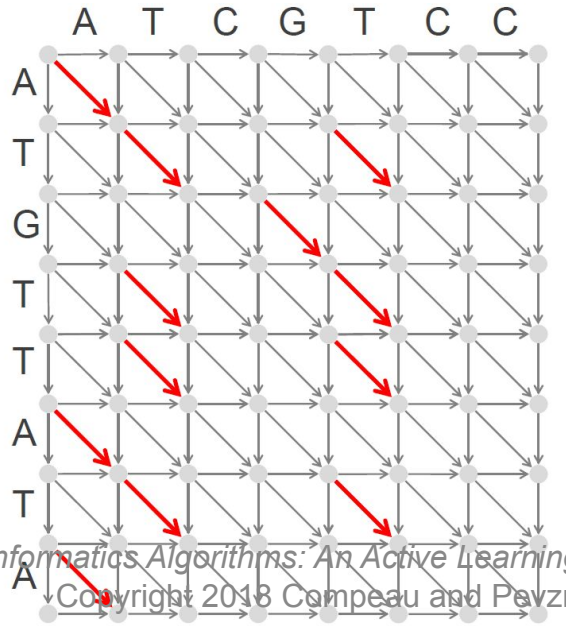
4 + 2



Dynamic Programming Recurrence for the Alignment Graph

$s_{i,j}$: the length of a longest path from (0,0) to (i,j)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge "}\downarrow\text{" into } (i,j) \\ s_{i,j-1} + \text{weight of edge "}\rightarrow\text{" into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge "}\searrow\text{" into } (i,j) \end{cases}$$

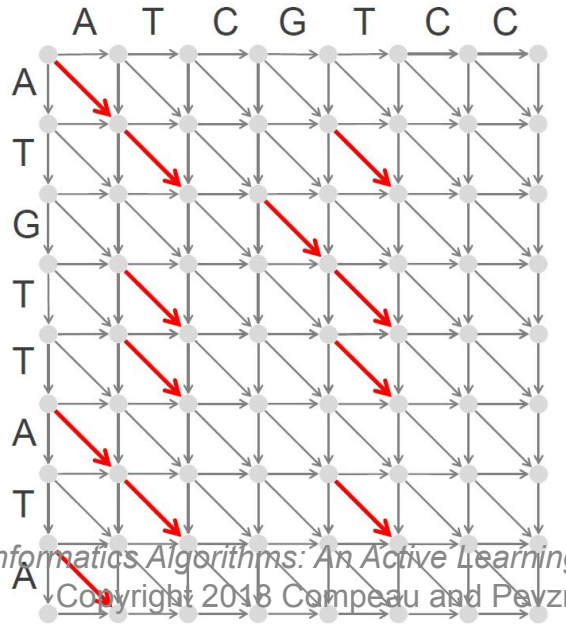


red edges ↘ – weight 1
other edges – weight 0

Dynamic Programming Recurrence for the Longest Common Subsequence Problem

$s_{i,j}$: the length of a longest path from (0,0) to (i,j)

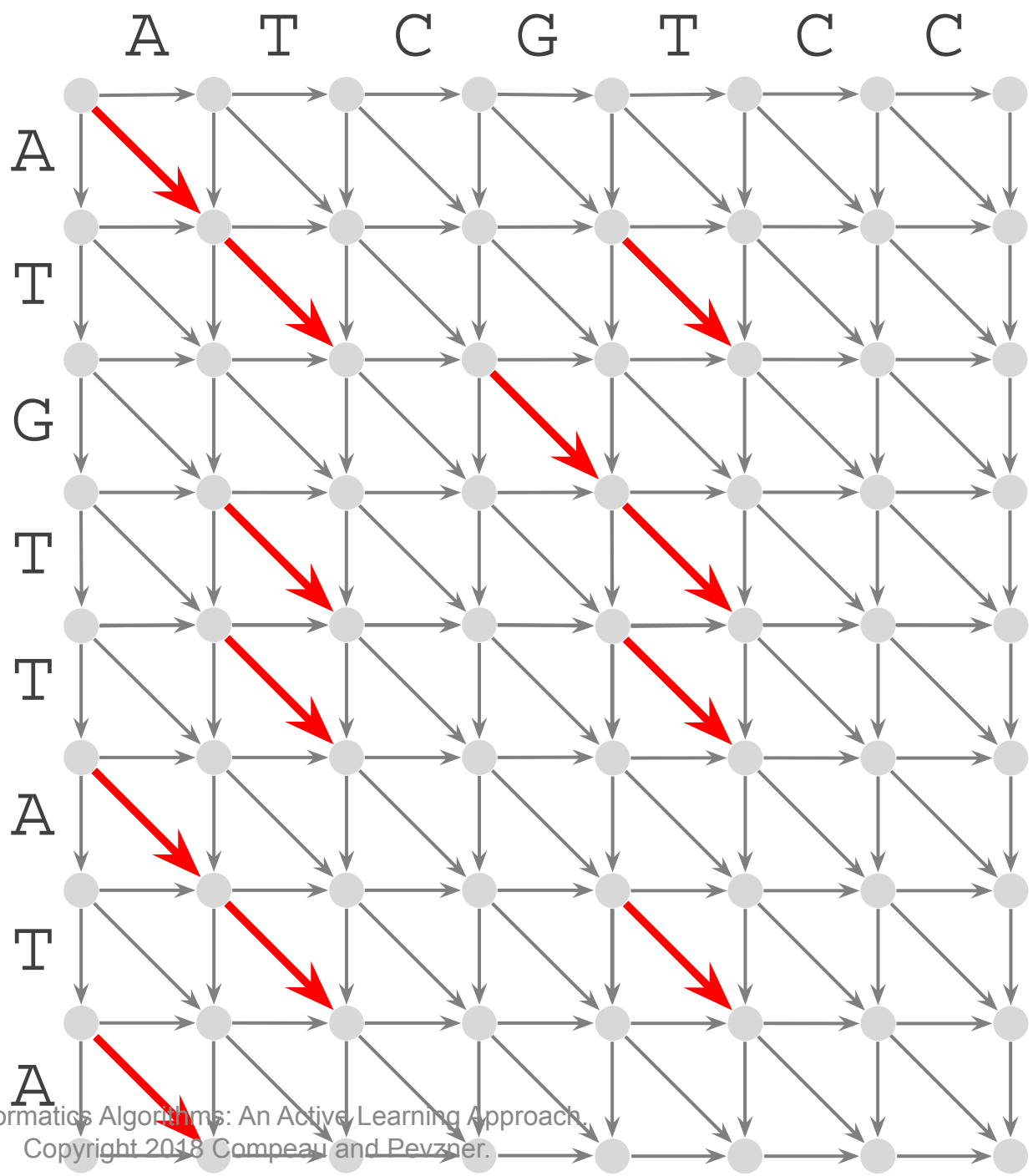
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$$



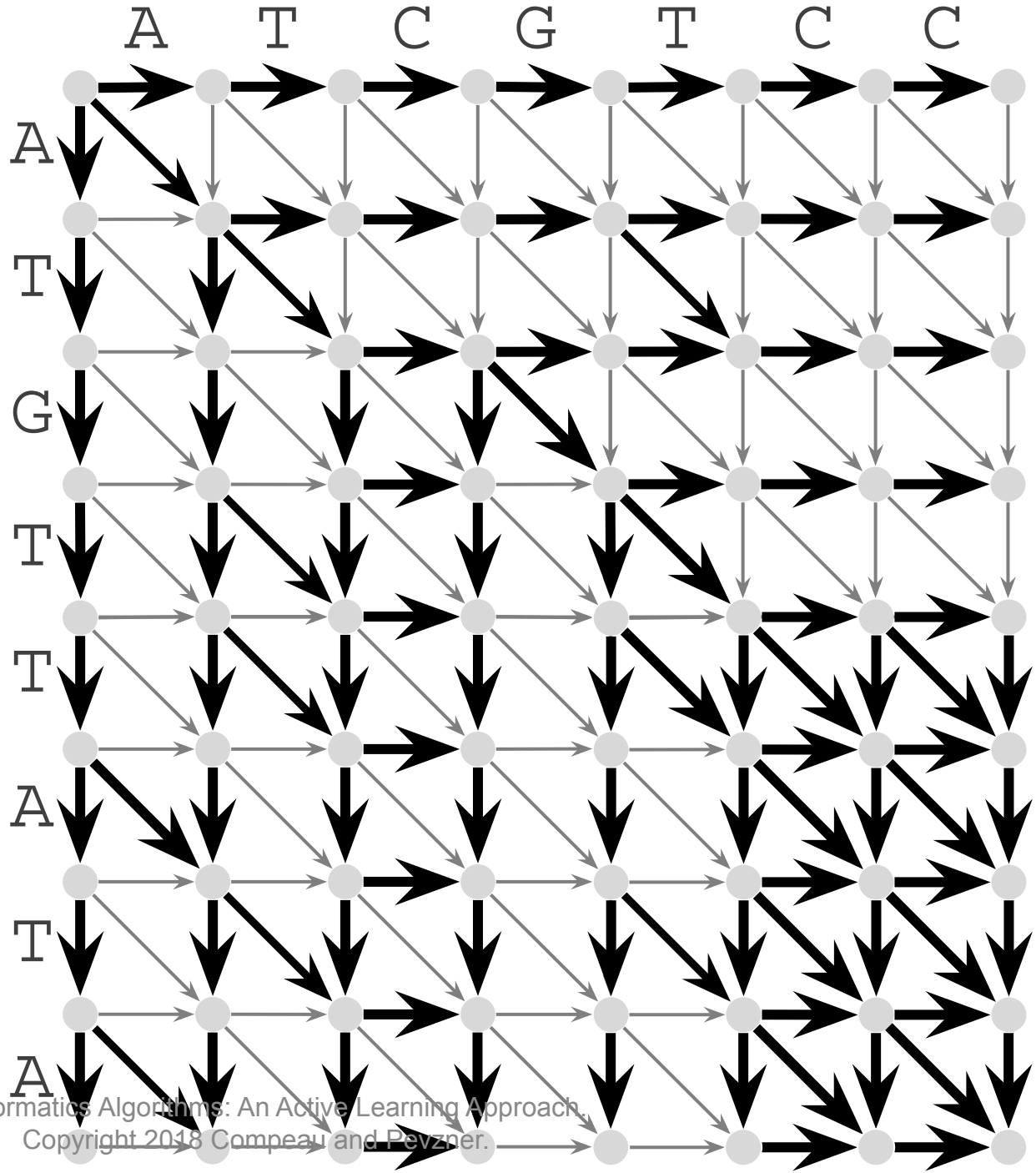
red edges ↘ – weight 1
other edges – weight 0

backtracking pointers for the Longest Common Subsequence

red edges ↘ – weight 1
other edges – weight 0



backtracking pointers
for the Longest
Common Subsequence

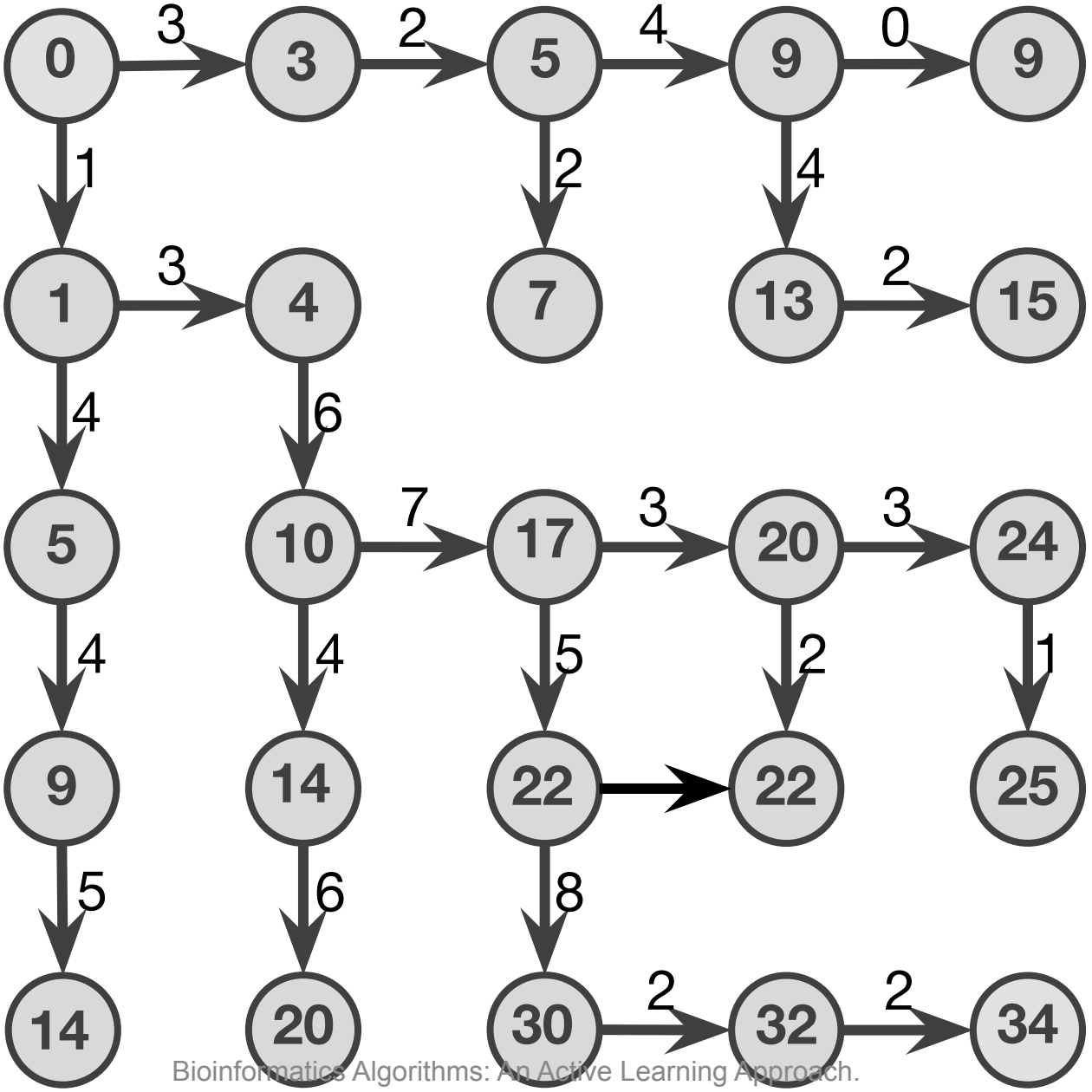


Computing Backtracking Pointers

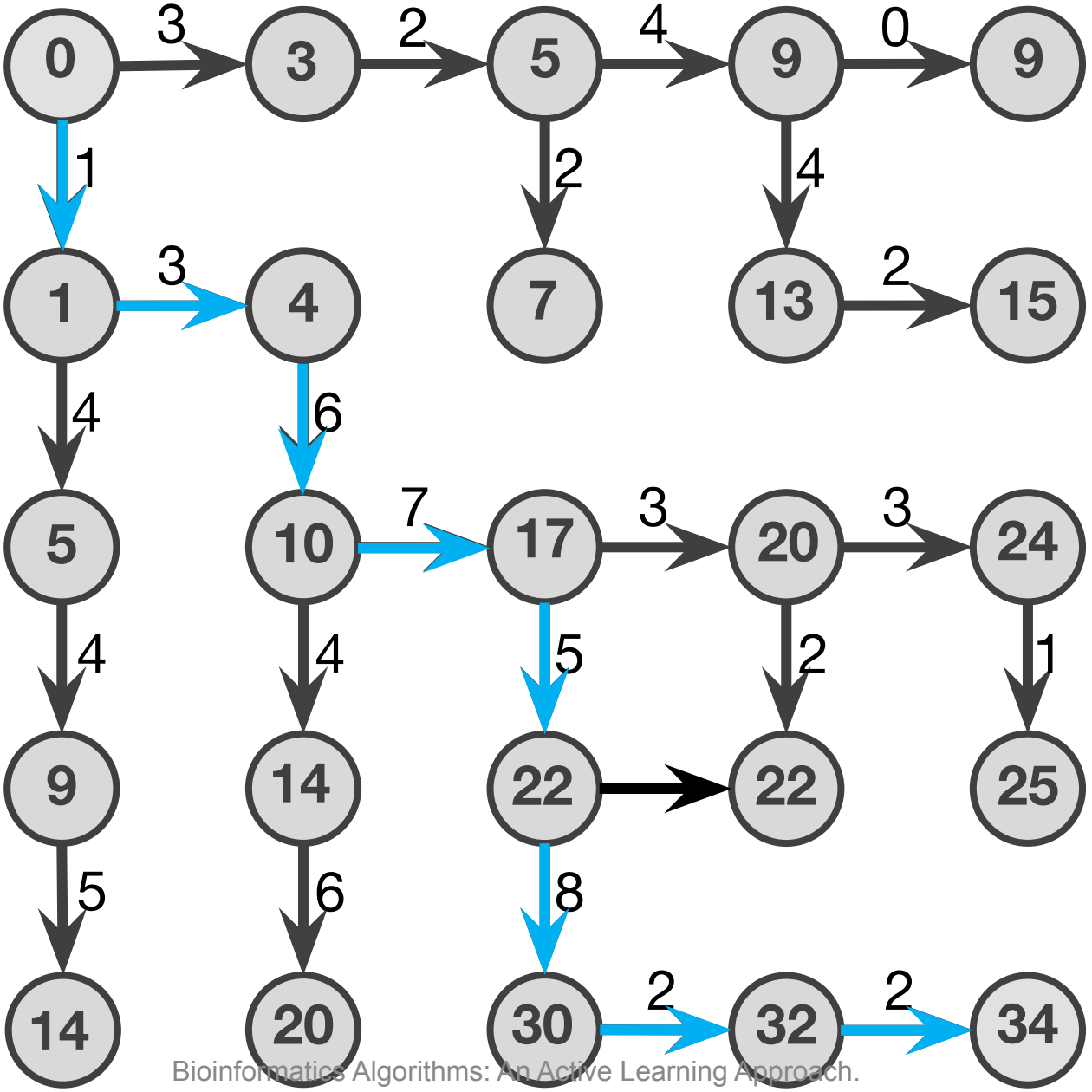
$$s_{i,j} \leftarrow \max\{ s_{i,j-1}+0,\ s_{i-1,j}+0,\ s_{i-1,j-1}+1, \text{ if } v_i=w_j$$

$$backtrack_{i,j} \leftarrow \{ \text{“}\rightarrow\text{”, if } s_{i,j}=s_{i,j-1},\ \text{“}\downarrow\text{”, if } s_{i,j}=s_{i-1,j},\ \text{“}\searrow\text{”, if } s_{i,j}=s_{i-1,j-1}+1$$

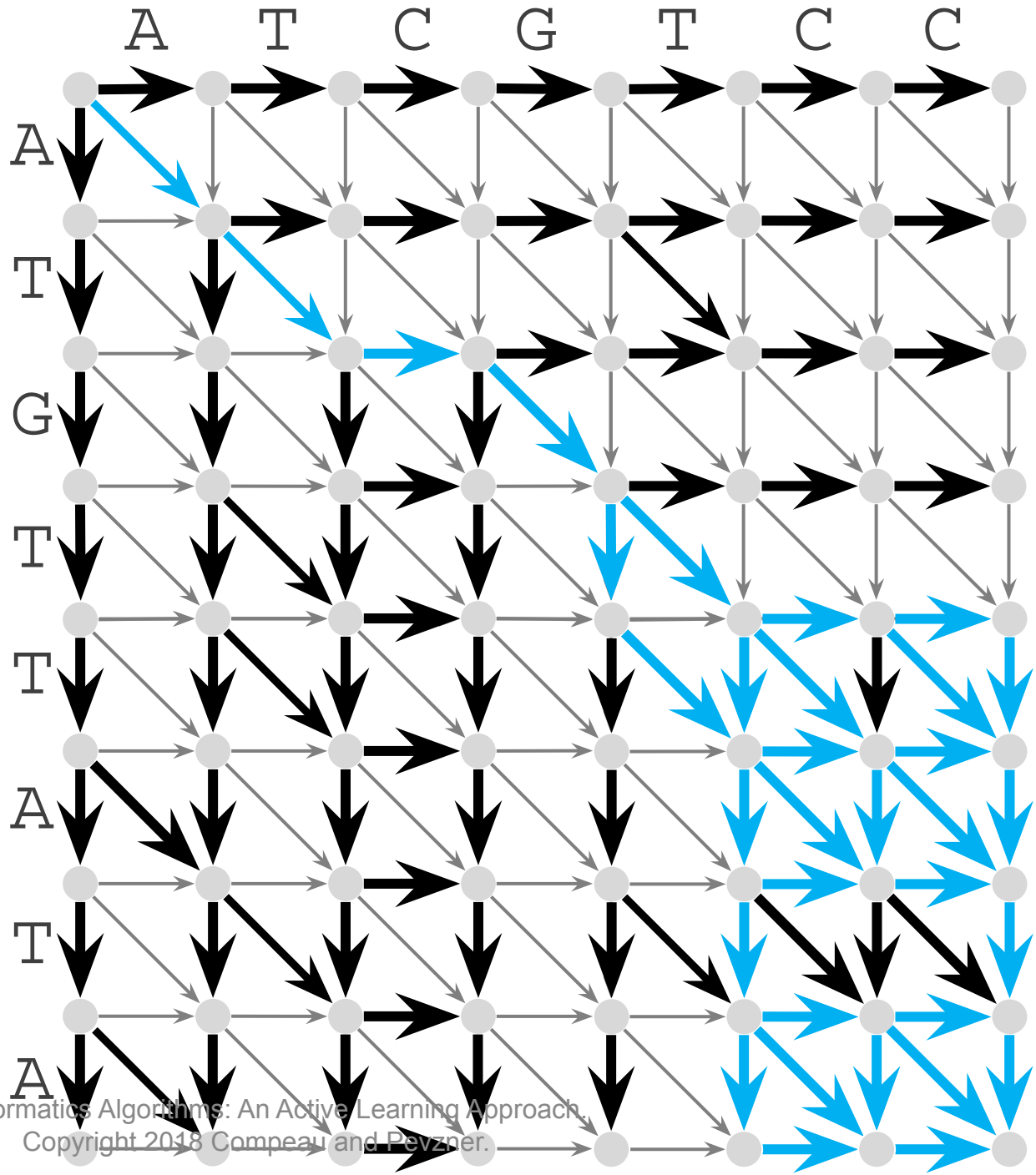
Why did we store the **backtracking pointers?**



What is the optimal alignment path?



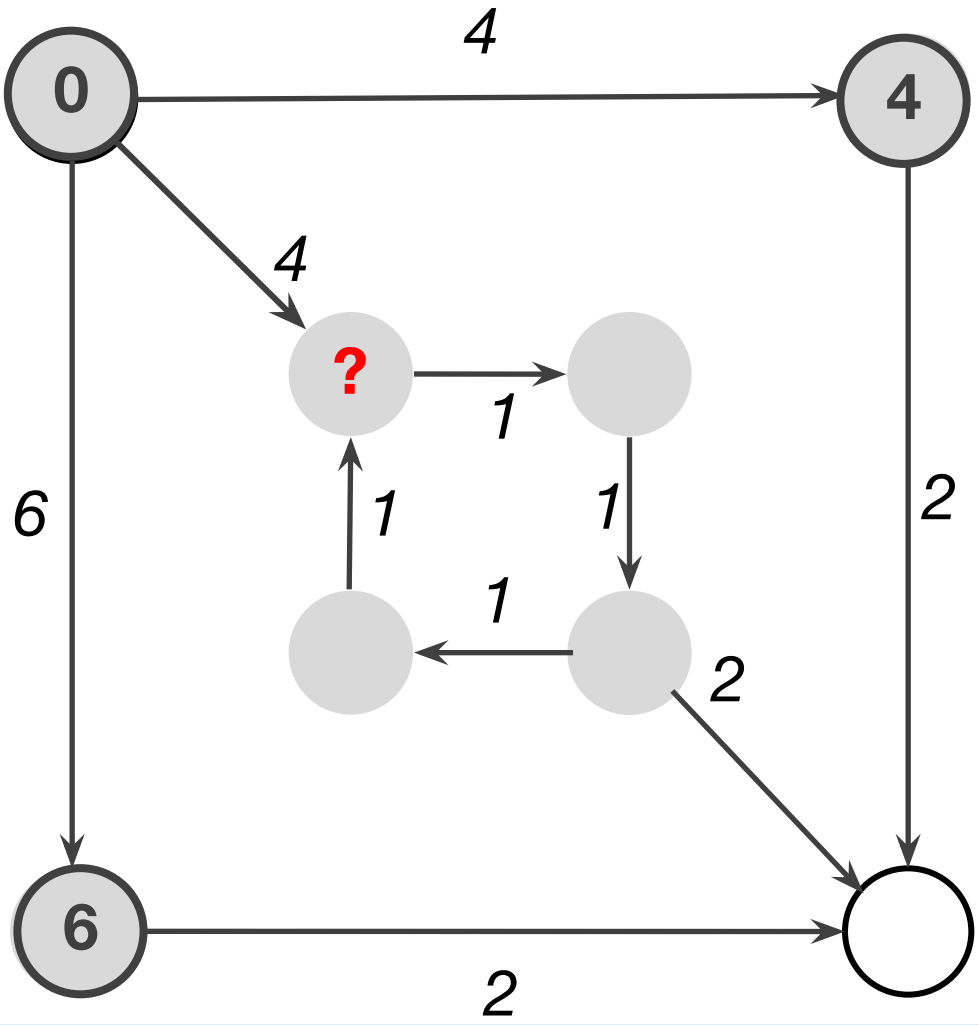
backtracking pointers
for the Longest
Common Subsequence



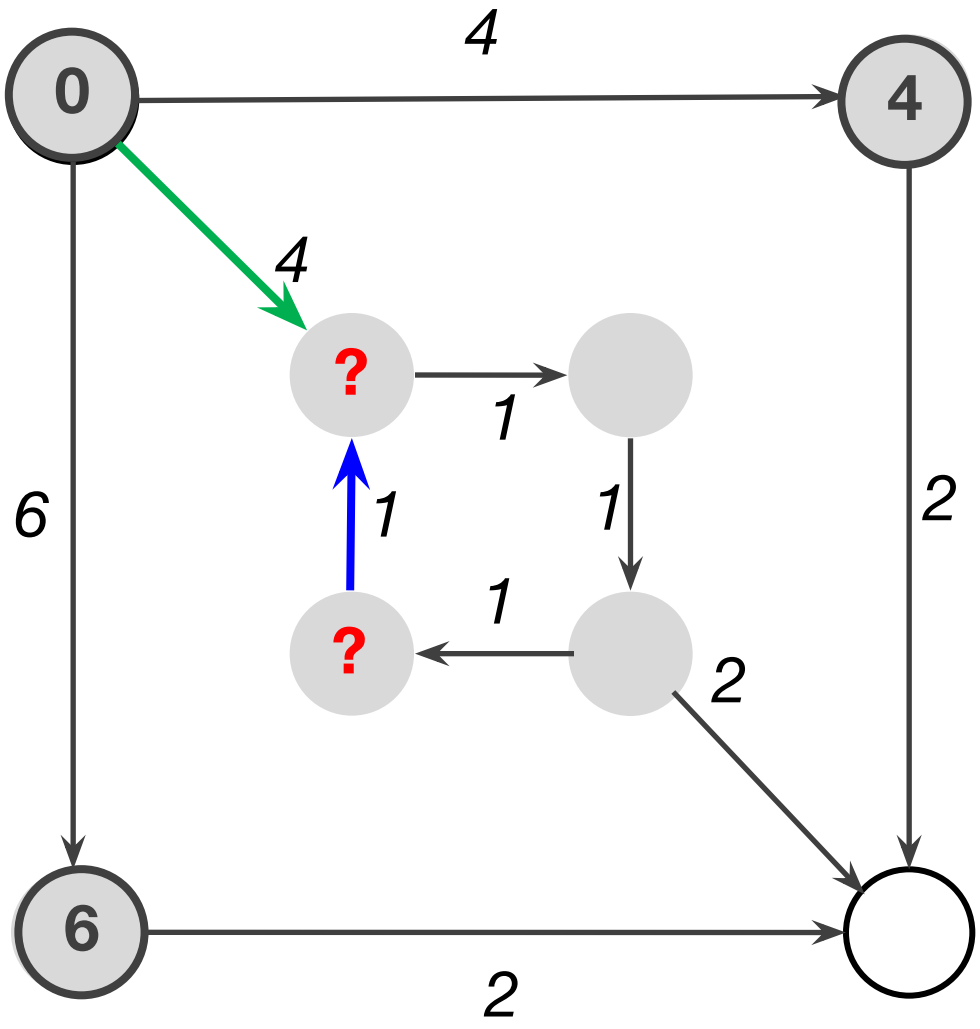
Using Backtracking Pointers to Compute LCS

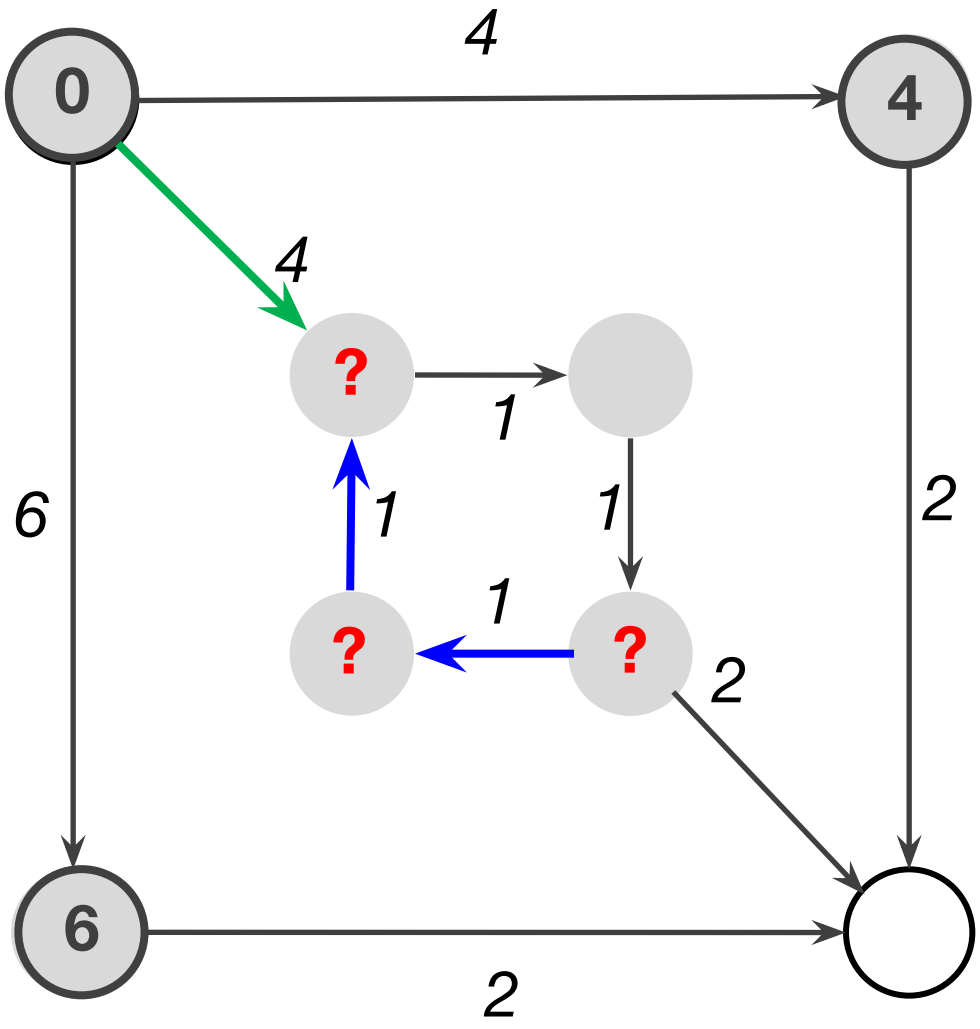
```
OutputLCS (backtrack, v, i, j)
  if i = 0 or j = 0
    return
  if backtrack_{i,j} = "→"
      OutputLCS (backtrack, v, i, j-1)
    else if backtrack_{i,j} = "↓"
      OutputLCS (backtrack, v, i-1, j)
    else
      OutputLCS (backtrack, v, i-1, j-1)
      output v_i
```

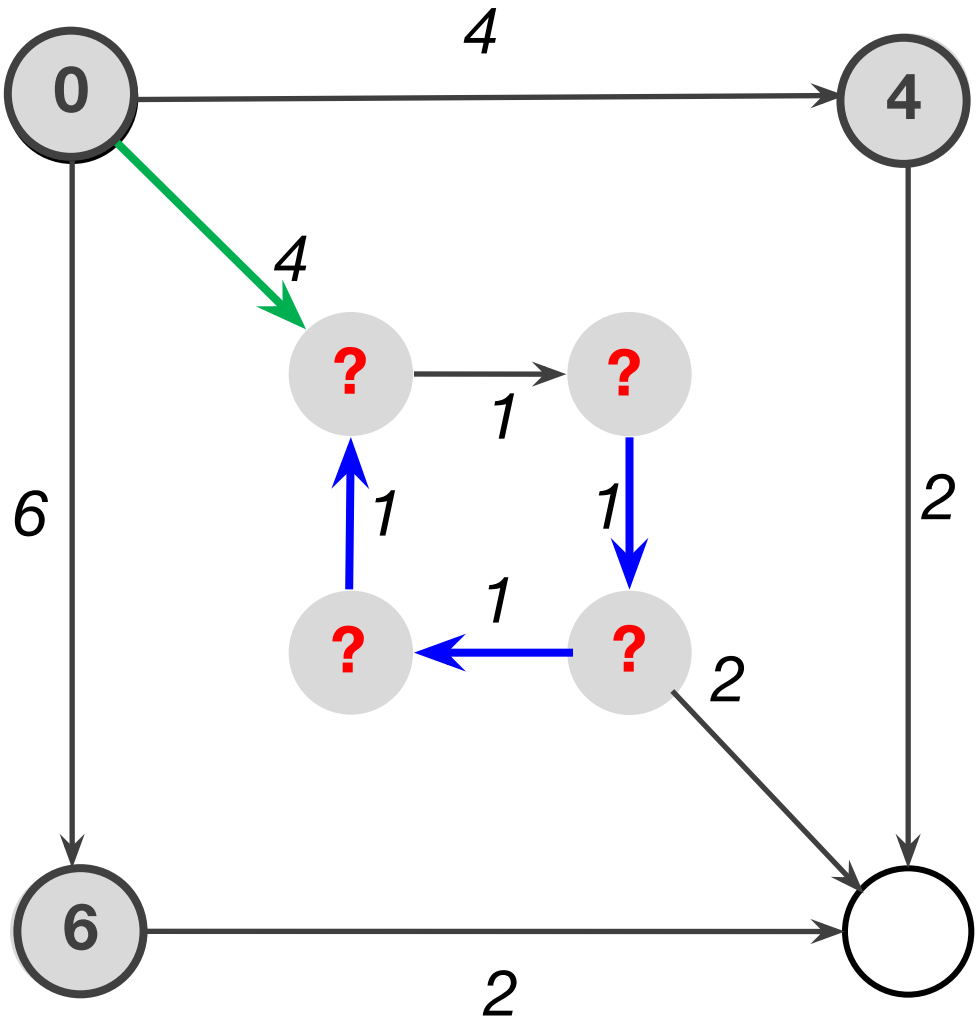
Computing Scores of ALL Predecessors



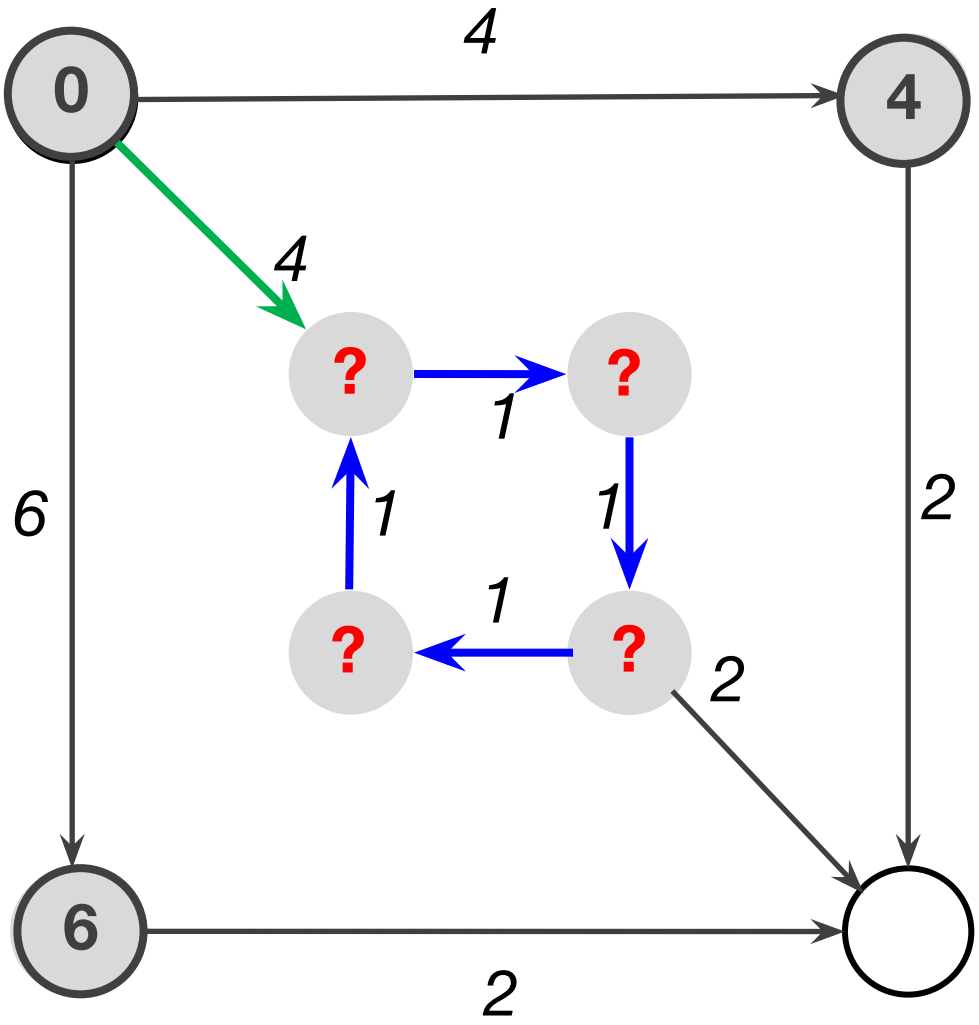
$s_a = \max_{\text{ALL predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$





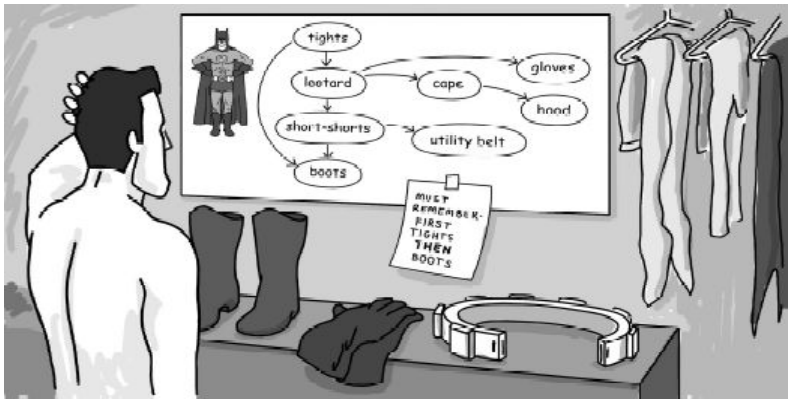


A Vicious Cycle



In What Order Should We Explore Nodes of the Graph?

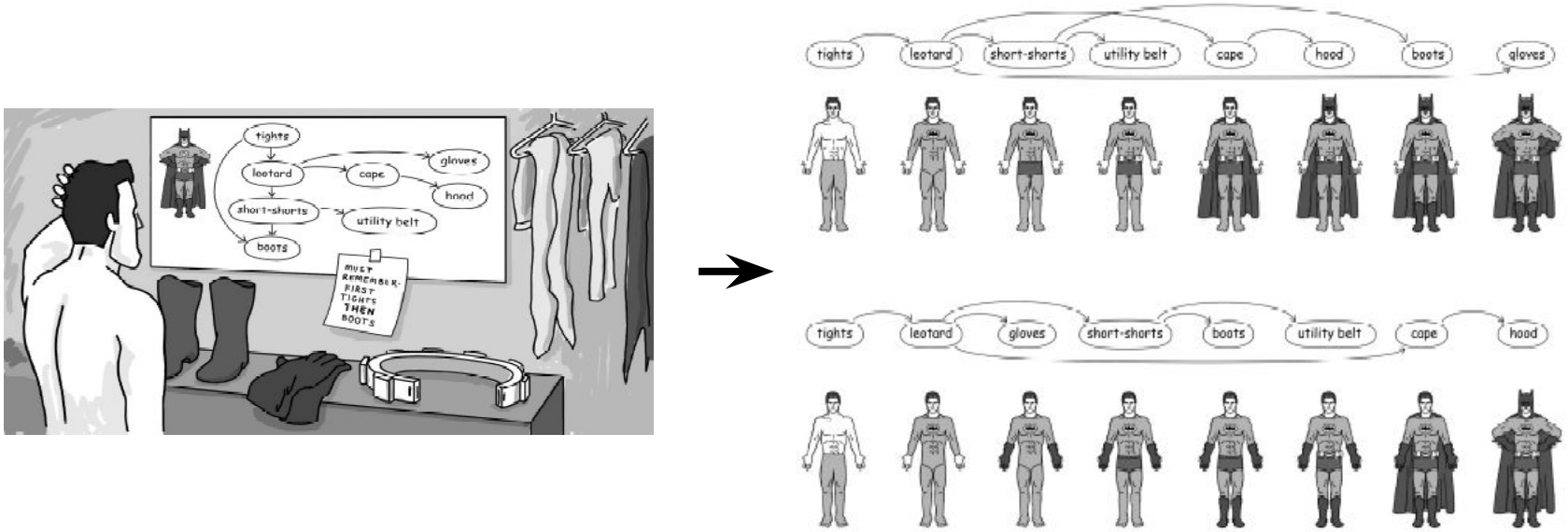
$s_a = \max_{\textbf{ALL}\ \text{predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$



- By the time a node is analyzed, the scores of all its predecessors should already be computed.
- If the graph has a **directed cycle**, this condition is impossible to satisfy.
- **Directed Acyclic Graph** (**DAG**): a graph without directed cycles.

Topological Ordering

- **Topological Ordering**: Ordering of nodes of a DAG on a line such that all edges go from left to right.



- **Theorem**: Every DAG has a topological ordering.

LongestPath

LongestPath(*Graph, source, sink*)
  for each node *a* in *Graph*
    $s_a \leftarrow$ -infinity
  $s_{source} \leftarrow 0$
  topologically order *Graph*
  for each node *a* (from *source* to *sink* in topological order)
    $s_a \leftarrow \max_{\text{all predecessors } b \text{ of node } a}\{s_b + \text{weight of edge from } b \text{ to } a\}$
  return s_{sink}

How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- **From Global to Local Alignment**
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

Current (Primitive) Scoring

#matches

Mismatches and Indel Penalties

#matches $-\ \mu \cdot$ #mismatches $-\ \sigma \cdot$ #indels

```
A T - G T T A T A
A T C G T - C - C
+1+1-2+1+1-2-3-2-3=-7
```

	A	C	G	T	-
A	+1	$-\mu$	$-\mu$	$-\mu$	$-\sigma$
C	$-\mu$	+1	$-\mu$	$-\mu$	$-\sigma$
G	$-\mu$	$-\mu$	+1	$-\mu$	$-\sigma$
T	$-\mu$	$-\mu$	$-\mu$	+1	$-\sigma$
-	$-\sigma$	$-\sigma$	$-\sigma$	$-\sigma$	

Scoring matrix

	A	C	G	T	-
A	+1	-3	-5	-1	-3
C	-4	+1	-3	-2	-3
G	-9	-7	+1	-1	-3
T	-3	-5	-8	+1	-4
-	-4	-2	-2	-1	

Even more general scoring matrix

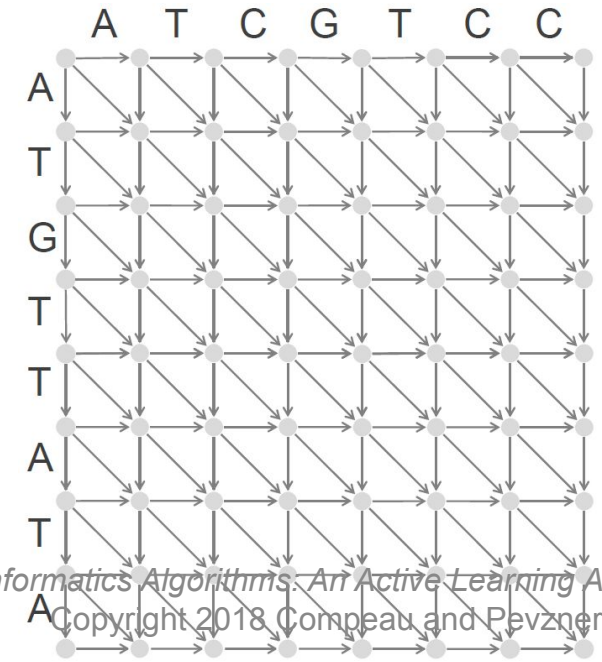
Scoring Matrices for Amino Acid Sequences

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C Cys	12																			
S Ser	0	2																		
T Thr	-2	1	3																	
P Pro	-3	1	0	6																
A Ala	-2	1	1	1	2															
G Gly	-3	1	0	-1	1	5														
N Asn	-4	1	0	-1	0	0	2													
D Asp	-5	0	0	-1	0	1	2	4												
E Glu	-5	0	0	-1	0	0	1	3	4											
Q Gln	-5	-1	-1	0	0	-1	1	2	2	4										
H His	-3	-1	-1	0	-1	-2	2	1	1	3	6									
R Arg	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6								
K Lys	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5							
M Met	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6						
I Ile	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5					
L Leu	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6				
V Val	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4			
F Phe	-4	-3	-3	-5	-5	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9		
Y Tyr	0	-3	-3	**-5**	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	**7**	10	
W Trp	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17

Y (Tyr) often mutates into F (score +7)
but rarely mutates into P (score -5)

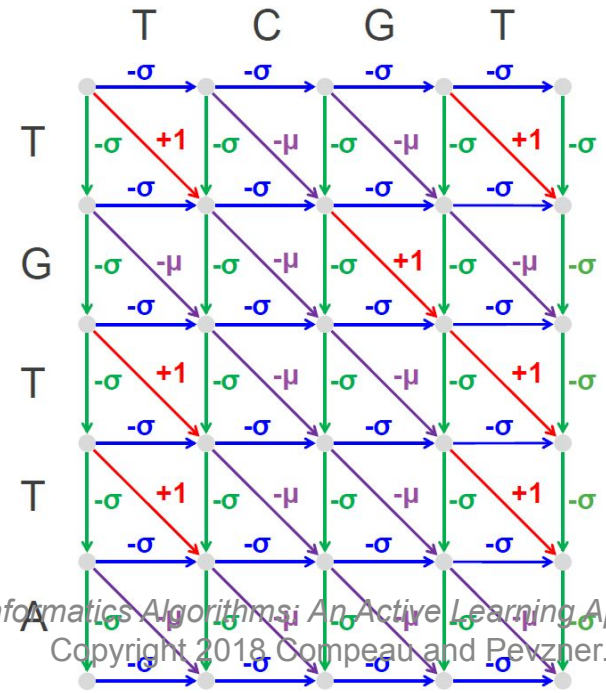
Dynamic Programming Recurrence for the Alignment Graph

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge "}\downarrow\text{" into } (i,j) \\ s_{i,j-1} + \text{weight of edge "}\rightarrow\text{" into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge "}\searrow\text{" into } (i,j) \end{cases}$$



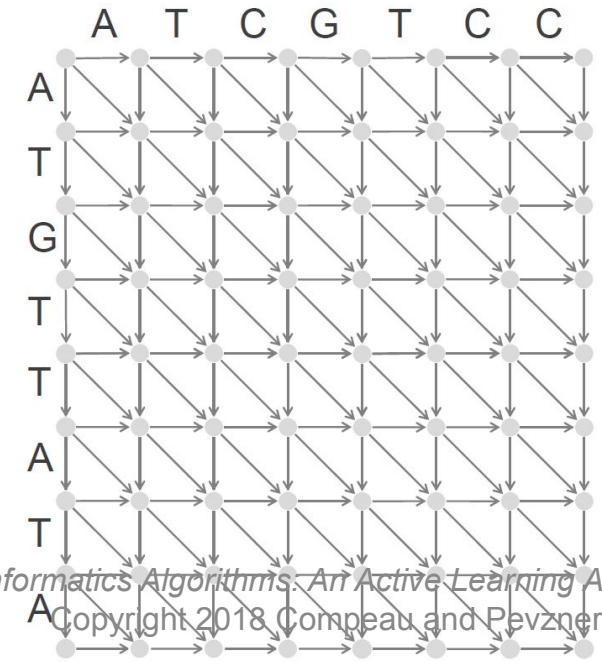
Dynamic Programming Recurrence for the Alignment Graph

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j \end{cases}$$



Dynamic Programming Recurrence for the Alignment Graph

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + score(v_i, -) \\ s_{i,j-1} + score(-, w_j) \\ s_{i-1,j-1} + score(v_i, w_j) \end{cases}$$



Global Alignment

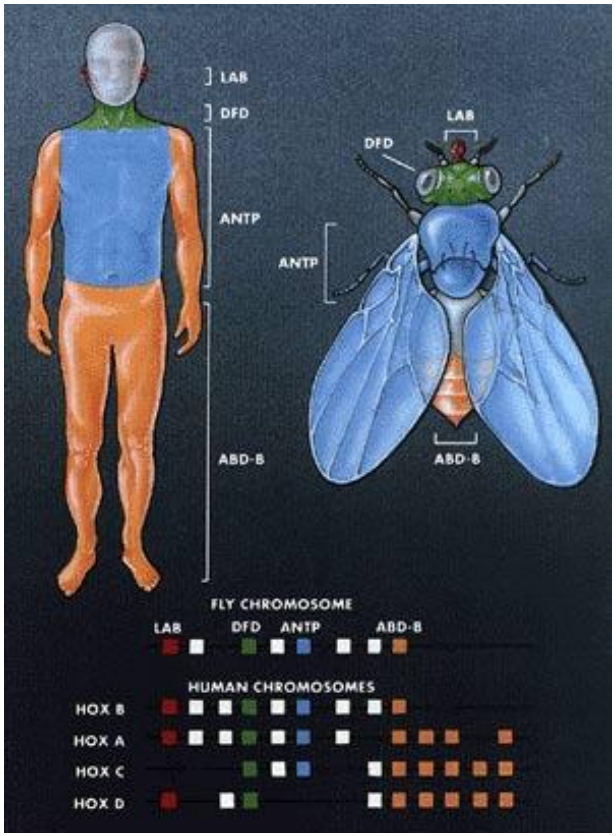
Global Alignment Problem: Find the highest-scoring alignment between two strings by using a scoring matrix.

- **Input:** Strings *v* and *w* as well as a matrix ***score***.
- **Output:** An alignment of *v* and *w* whose alignment score (as defined by the scoring matrix ***score***) is maximal among all possible alignments of *v* and *w*.

Homeobox Genes

- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.

- Homeobox genes have a short region called the **homeodomain** that is highly conserved among species.
  - A global alignment may not find the homeodomain because it would try to align the *entire* sequence.



Which Alignment is Better?

- Alignment 1: score = 22 (matches) - 20 (indels)=2.

```
GCC-C-AGT--TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT
```

- Alignment 2: score = 17 (matches) - 30 (indels)=-13.

```
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T-----
```

Which Alignment is Better?

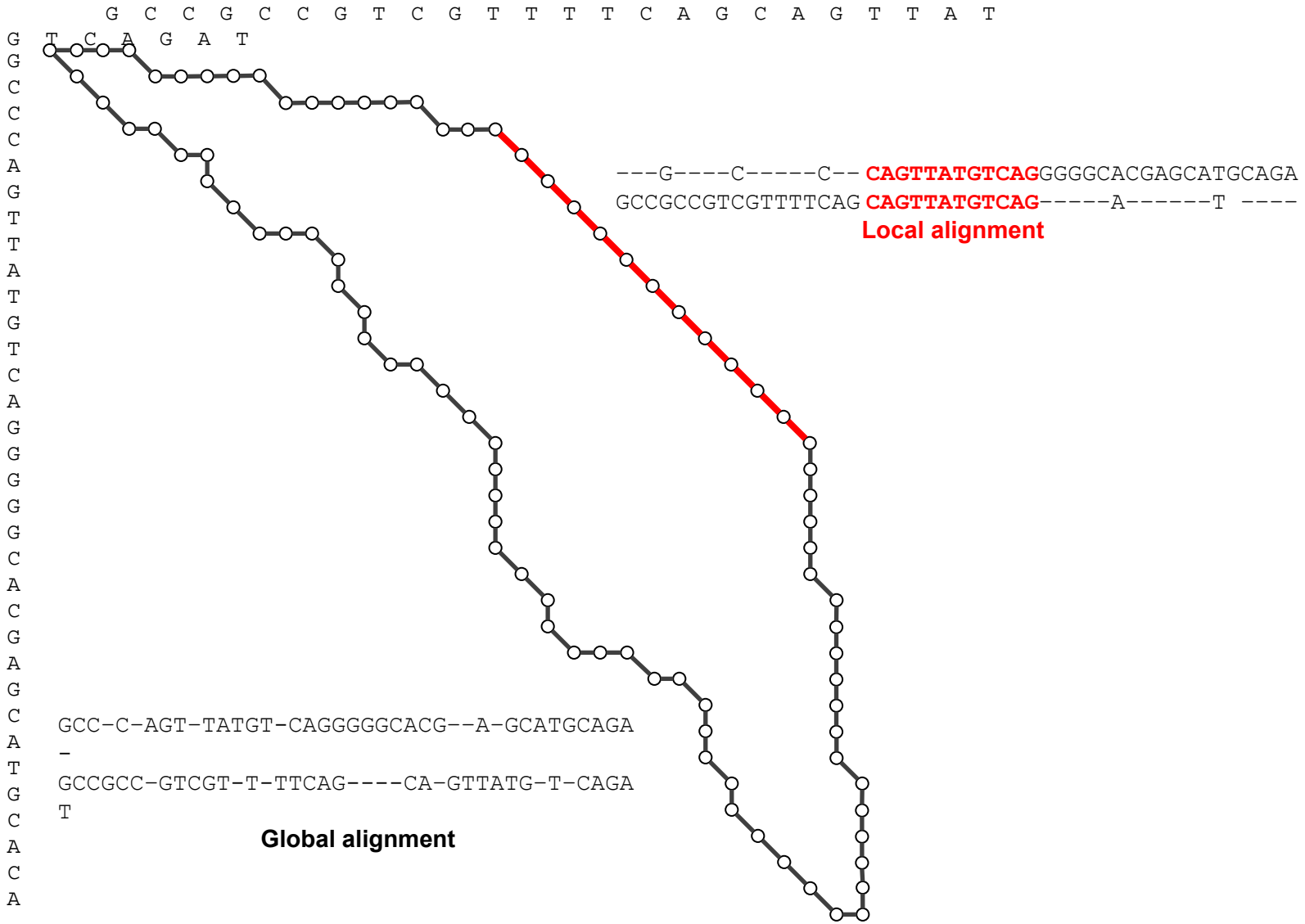
- Alignment 1: score = 22 (matches) - 20 (indels)=2.

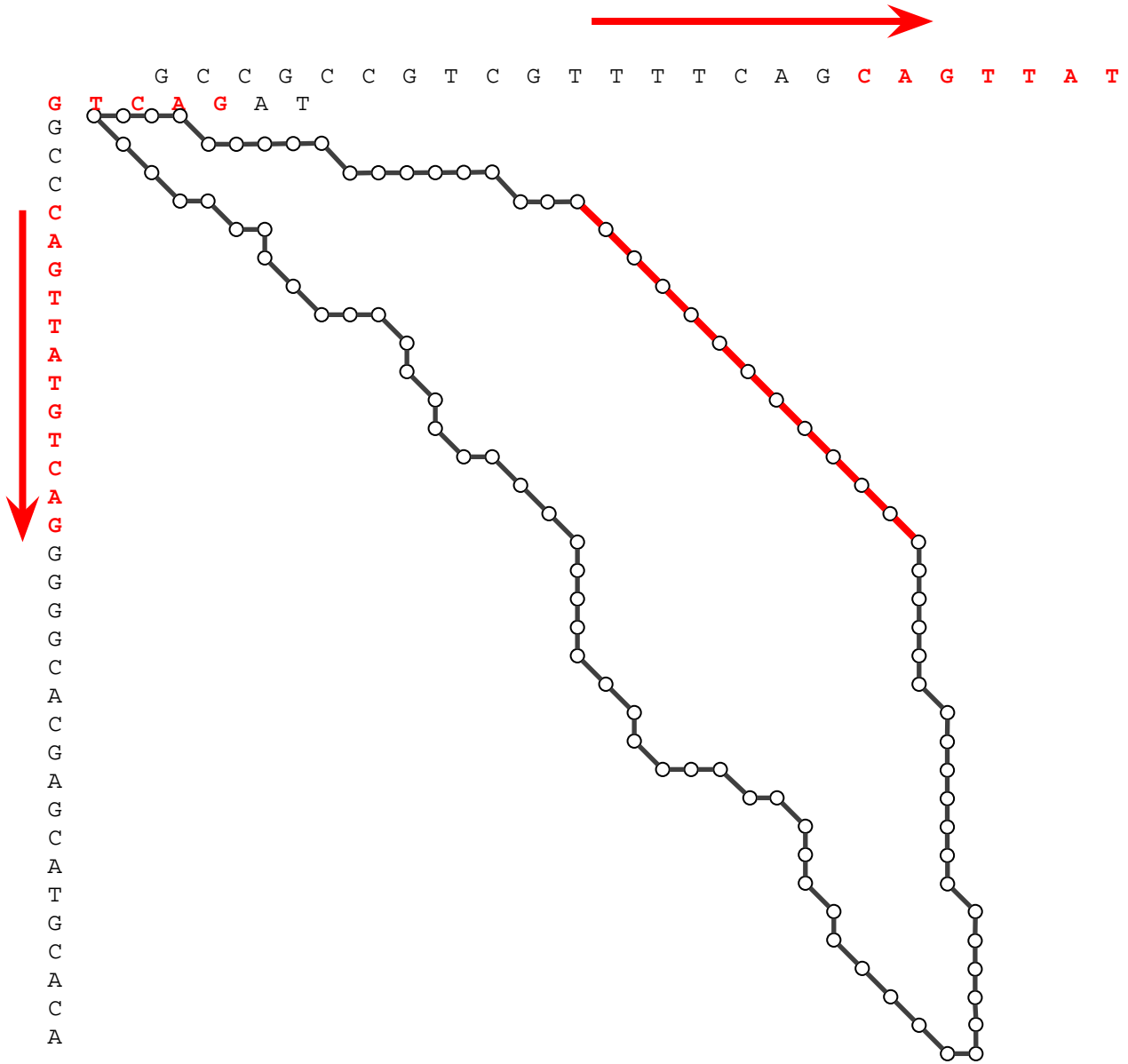
```
GCC-C-AGT--TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT
```

- Alignment 2: score = 17 (matches) - 30 (indels)=-13.

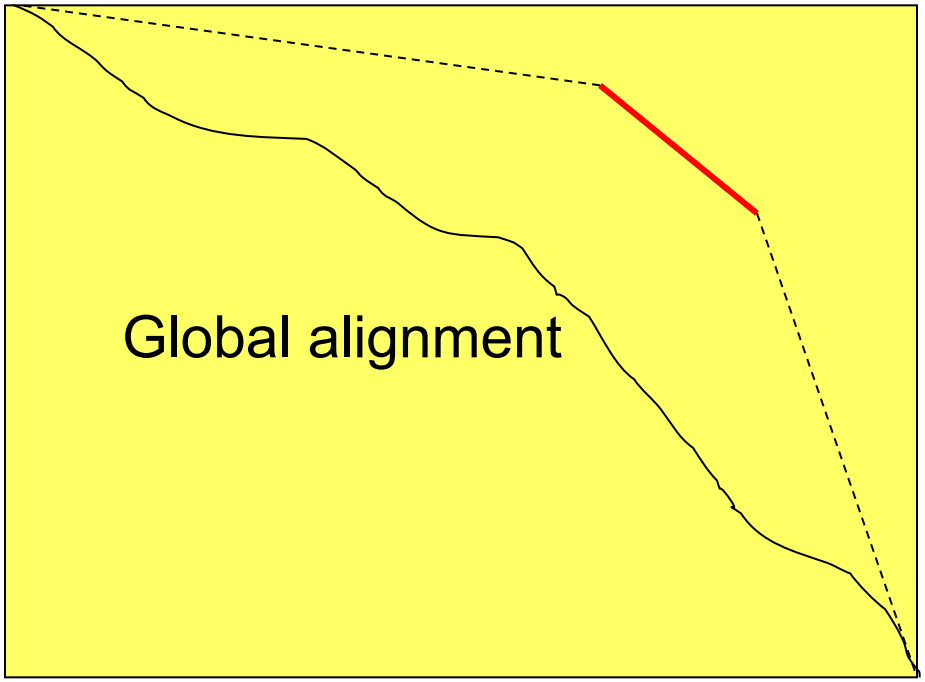
```
---G----C------C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T-----
```

local alignment

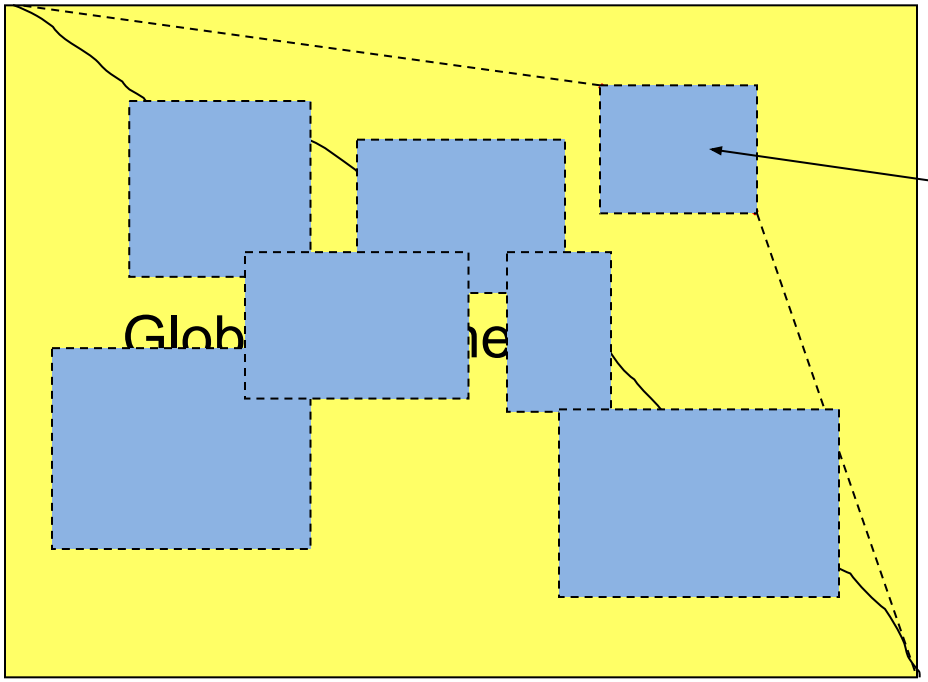




Local Alignment



Local Alignment= Global Alignment in a Subrectangle



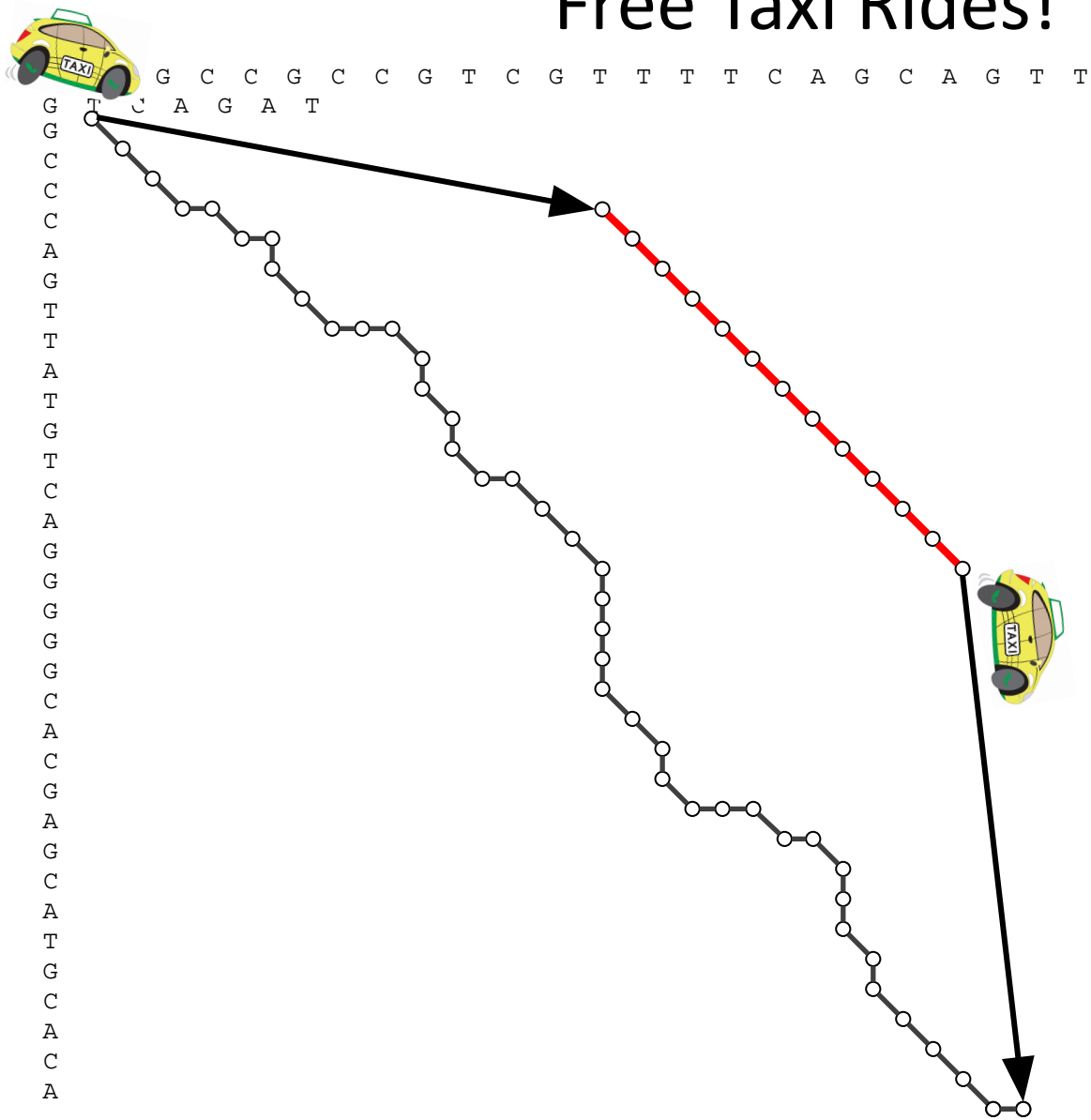
Compute a Global Alignment within each rectangle to get a Local Alignment

Local Alignment Problem

Local Alignment Problem: Find the highest-scoring local alignment between two strings.

- **Input:** Strings *v* and *w* as well as a matrix ***score***.
- **Output:** Substrings of *v* and *w* whose global alignment (as defined by the matrix ***score***), is maximal among all global alignments of all substrings of *v* and *w*.

Free Taxi Rides!

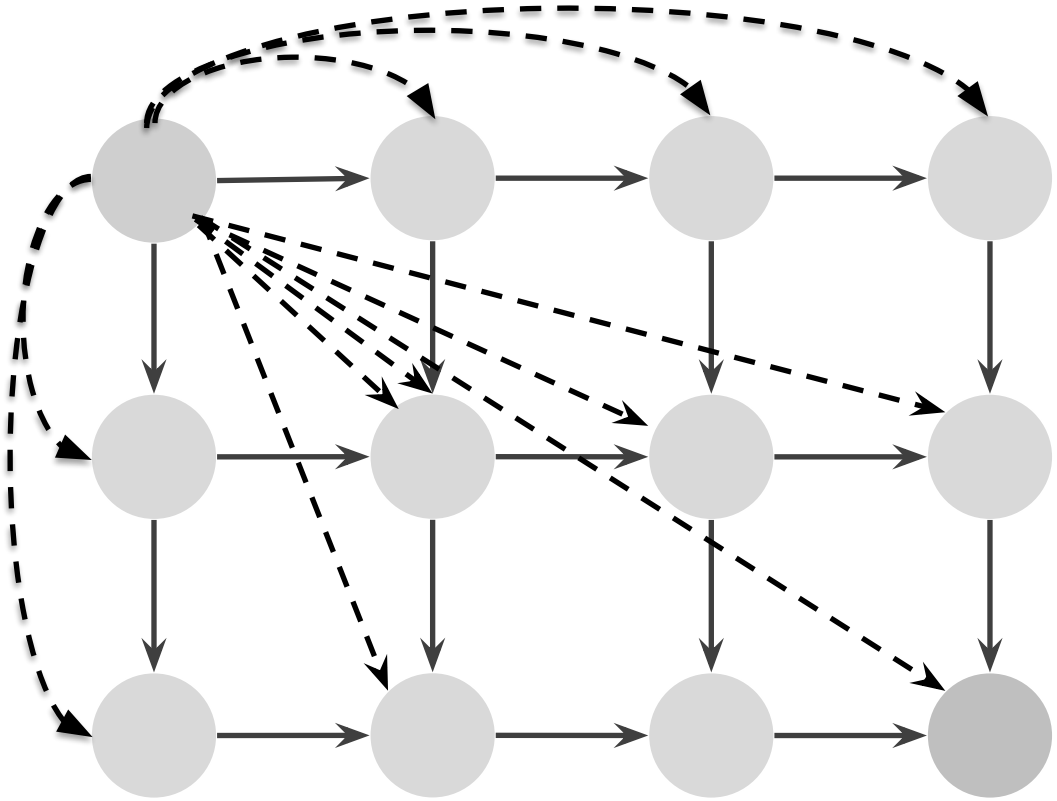


GCC-C-AGT-TATGT-CAGGGGGCACG--A-GCATGCACA
-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGA
T

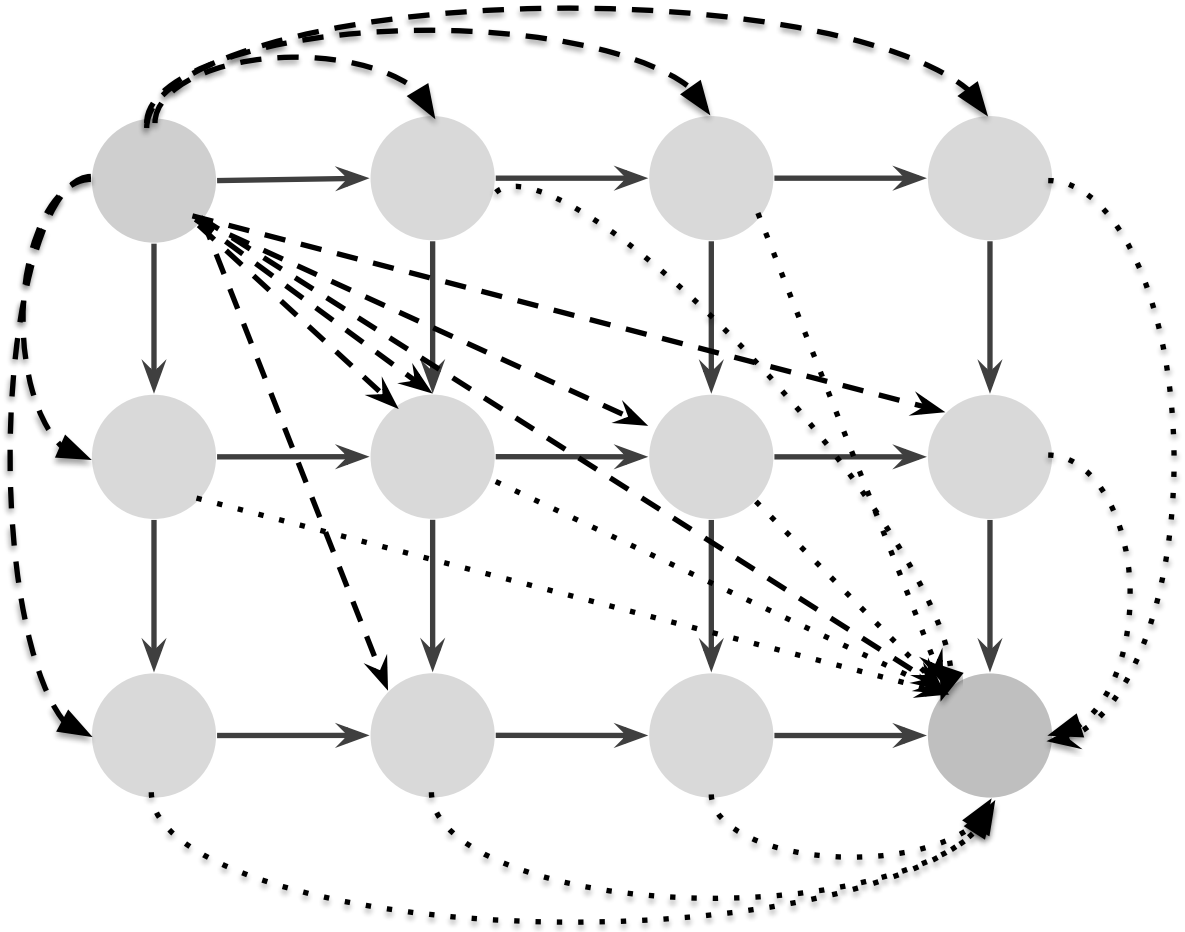
---G----C-----C-- **CAGTTATGTCAG**GGGGCACGAGCATGCACA
GCCGCCGTCGTTTTCAG**CAGTTATGTCAG**-----A------T ----

Local alignment

What Do Free Taxi Rides Mean in the Terms of the Alignment Graph?



Building Manhattan for the Local Alignment Problem

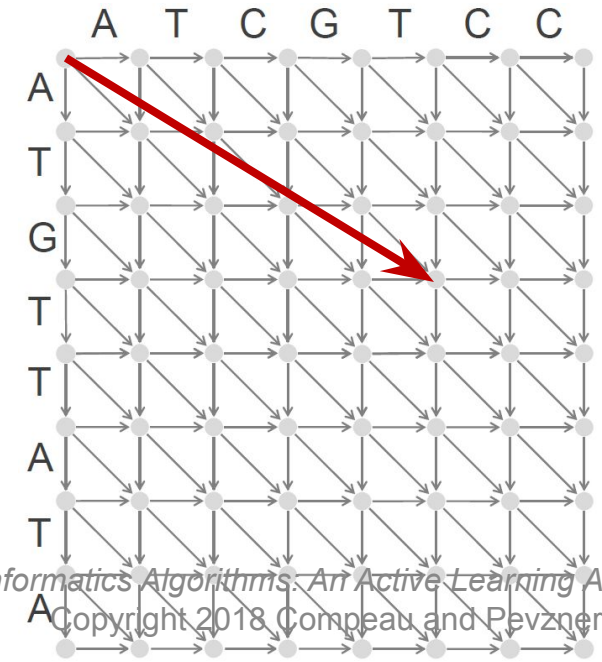


How many edges have we added?

Dynamic Programming for the Local Alignment

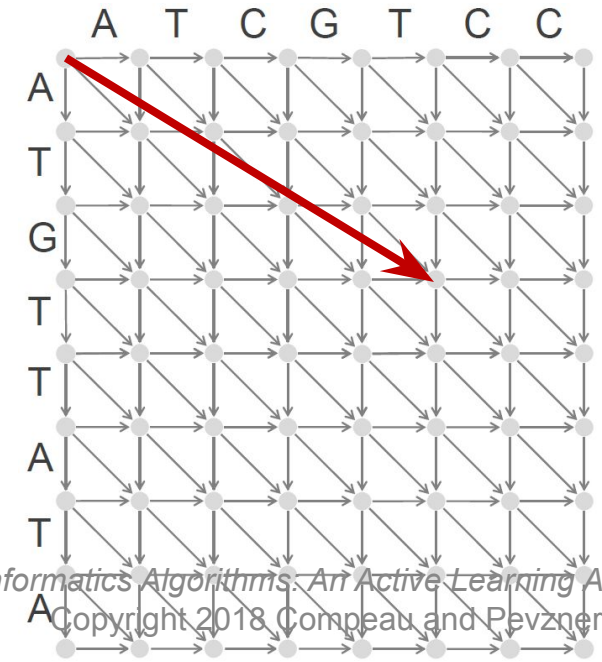
weight of edge from (0,0) to (i,j)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge "}\downarrow\text{" into } (i,j) \\ s_{i,j-1} + \text{weight of edge "}\rightarrow\text{" into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge "}\searrow\text{" into } (i,j) \end{cases}$$



Dynamic Programming for the Local Alignment

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \text{weight of edge “}\downarrow\text{” into } (i,j) \\ s_{i,j-1} + \text{weight of edge “}\rightarrow\text{” into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge “}\searrow\text{” into } (i,j) \end{cases}$$



How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to the Alignment Graph
- From Global to Local Alignment
- **Penalizing Insertions and Deletions in Sequence Alignment**
- Space-Efficient Sequence Alignment
- Multiple Sequence Alignment

Scoring Gaps

- We previously assigned a fixed penalty σ to each indel.
- However, this fixed penalty may be too severe for a series of 100 consecutive indels.
- A series of *k* indels often represents a single evolutionary event (**gap**) rather than *k* events:

two gaps (lower score)

```
GATCCAG
GA-C-AG
```

```
GATCCAG
GA--CAG
```

a single gap (higher score)

More Adequate Gap Penalties

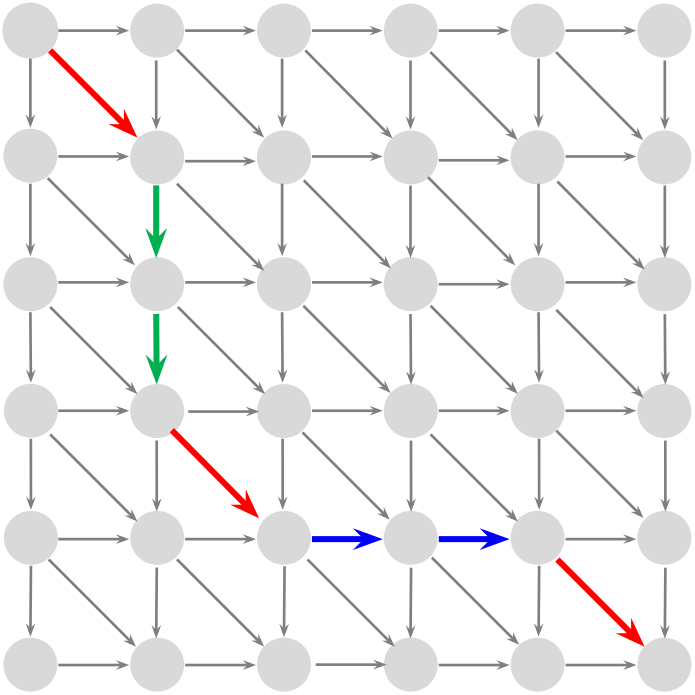
Affine gap penalty for a gap of length *k*: $\sigma+\varepsilon\cdot(k-1)$

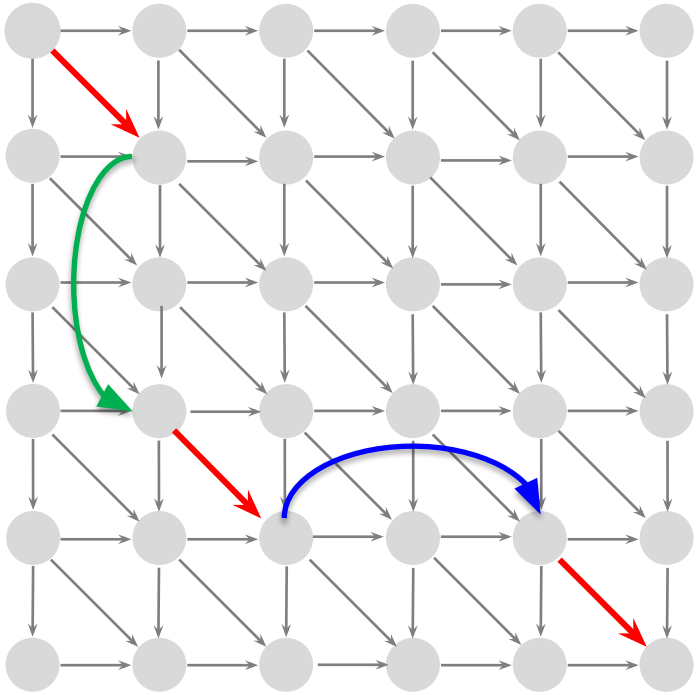
σ - the **gap opening penalty**

ε - the **gap extension penalty**

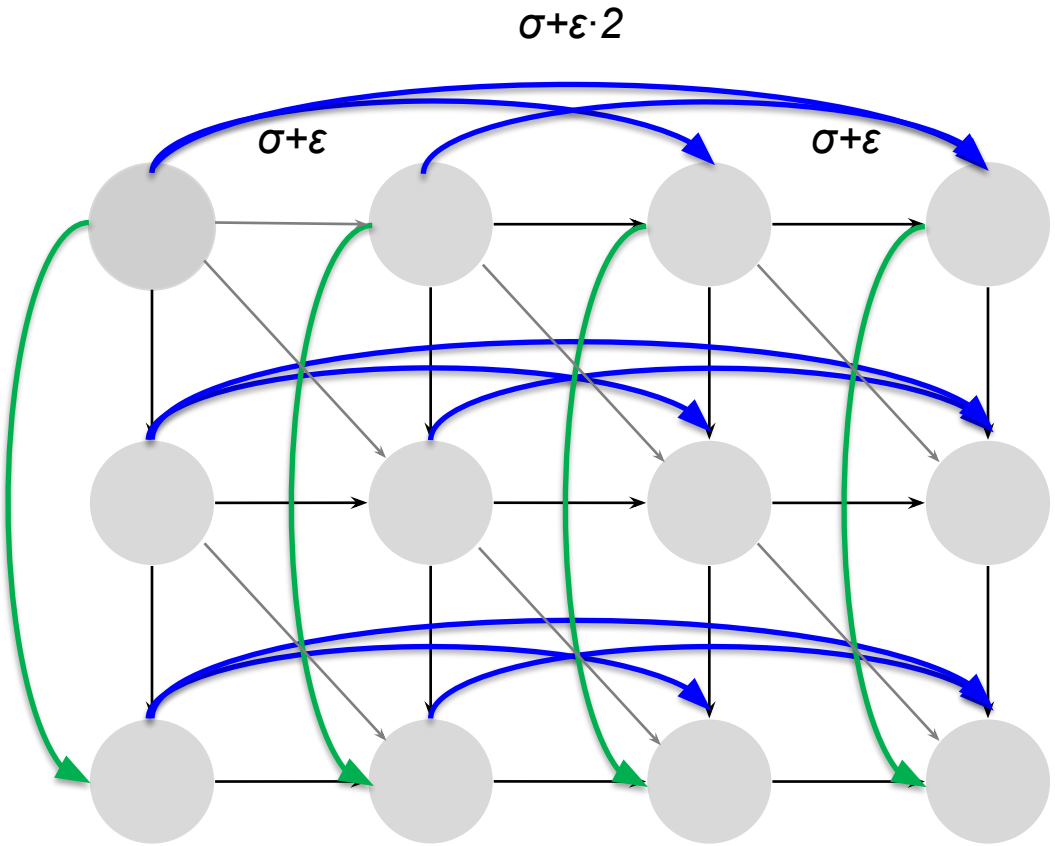
$\sigma > \varepsilon$, since starting a gap should be penalized more than extending it.

Modelling Affine Gap Penalties by Long Edges



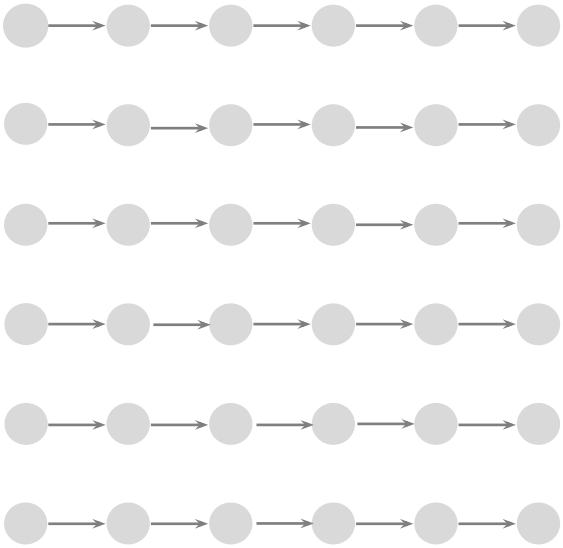


Building Manhattan with Affine Gap Penalties

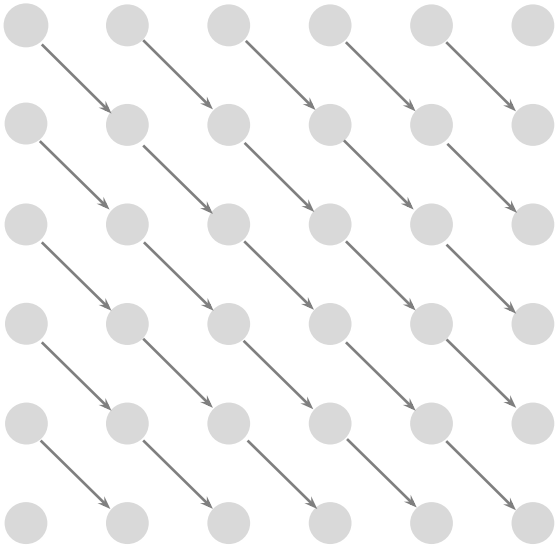


We have just added O(n^3) edges to the graph…

Building Manhattan on 3 levels

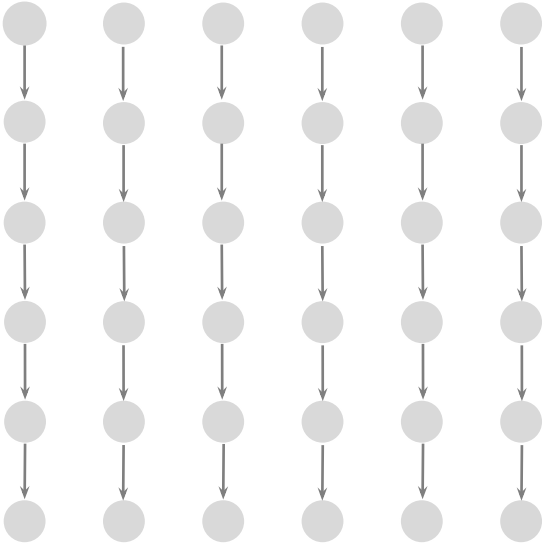


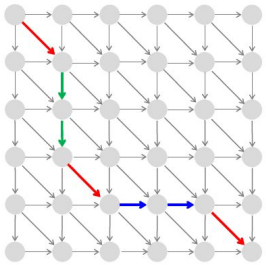
upper level
(deletions)



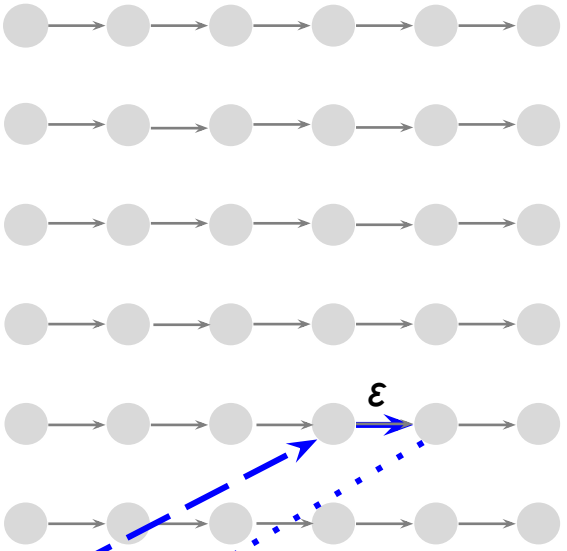
middle level
(matches/mismatches)

bottom level
(insertions)





How can we emulate this path in the 3-level Manhattan?



$lower_{i,j} = \max \{ lower_{i-1,j} - \varepsilon,\ middle_{i-1,j} - \sigma$

σ

ε

σ

0

$upper_{i,j} = \max \{ upper_{i,j-1} - \varepsilon,\ middle_{i,j-1} - \sigma$

0

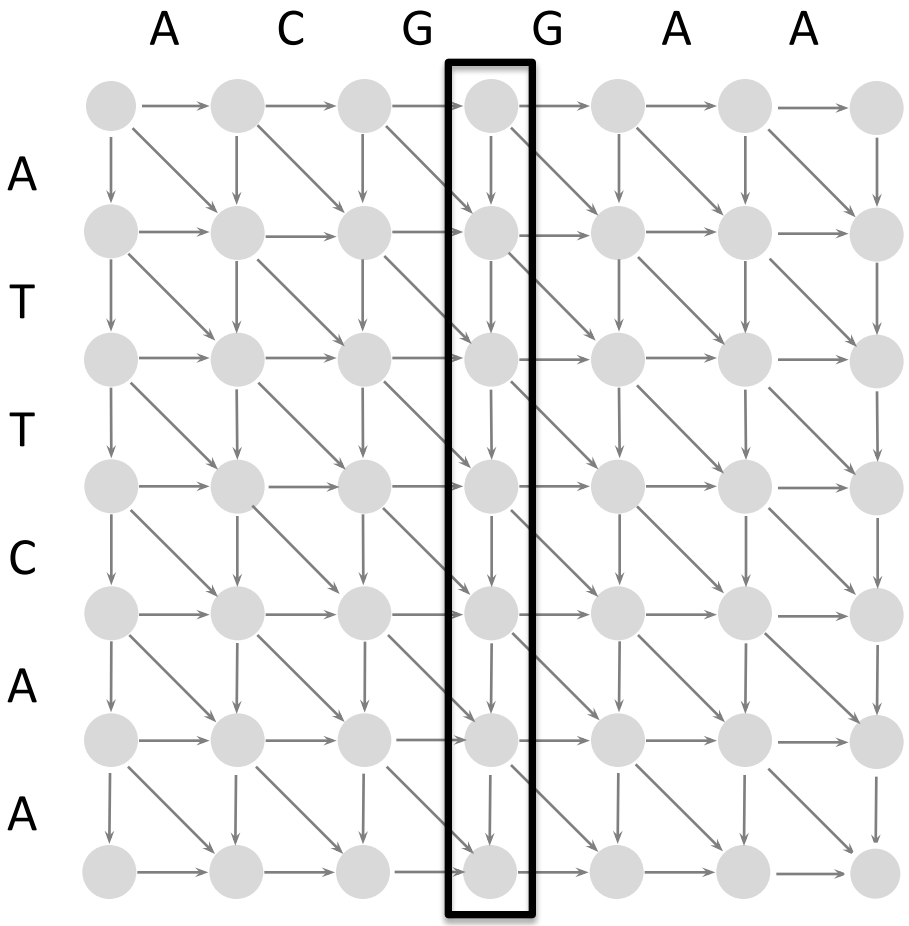
ε

$middle_{i,j} = \max \{ lower_{i,j},\ middle_{i-1,j-1} + score(v_i, w_j),\ upper_{i,j}$

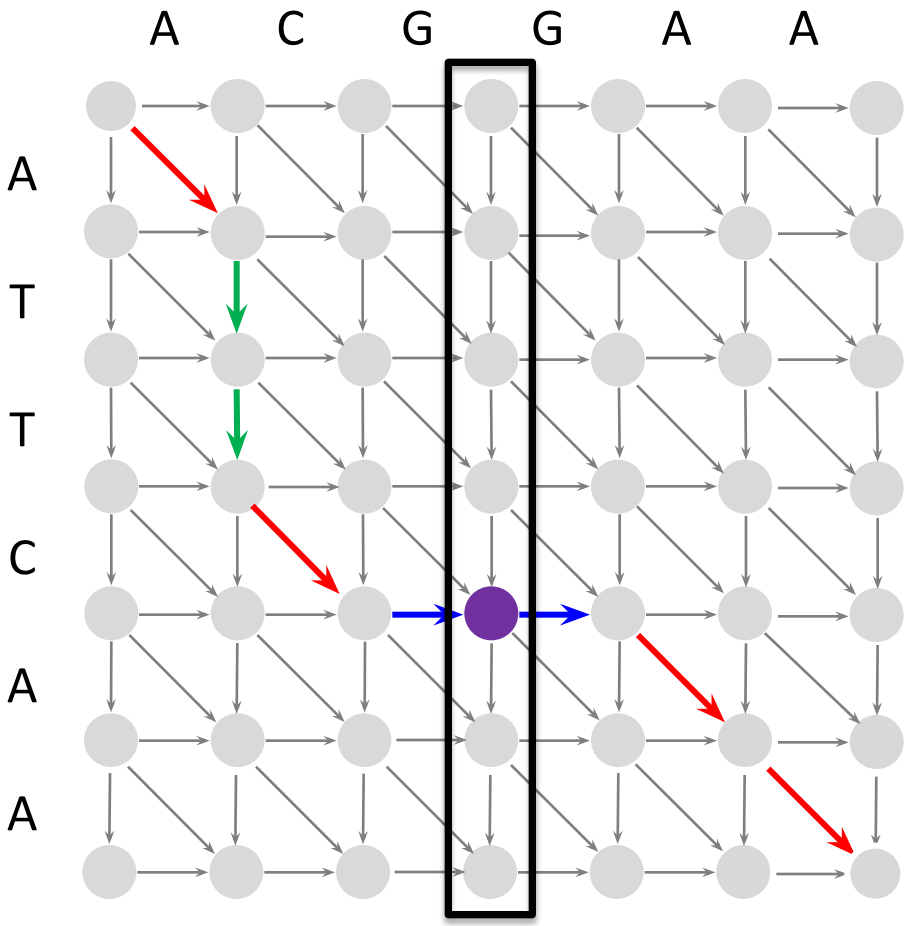
How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to an Arbitrary Directed Acyclic Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- **Space-Efficient Sequence Alignment**
- Multiple Sequence Alignment

Middle Column of the Alignment



Middle Node of the Alignment

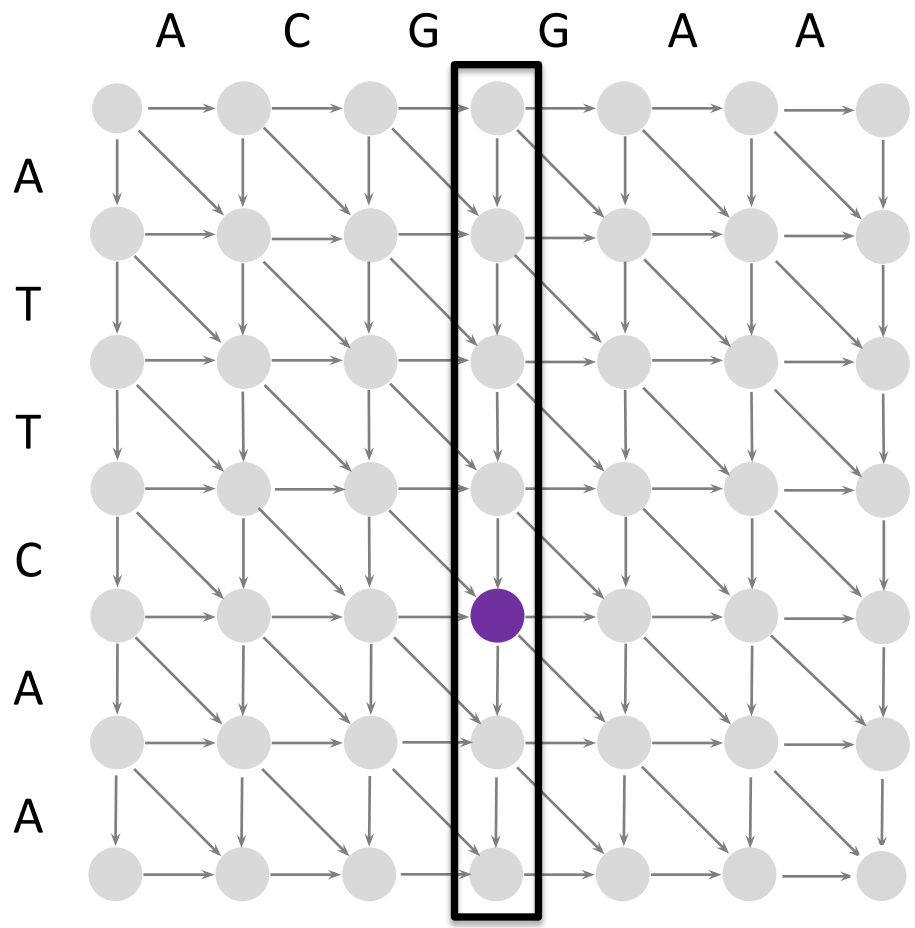


(a node where an optimal alignment path crosses the middle column)

Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

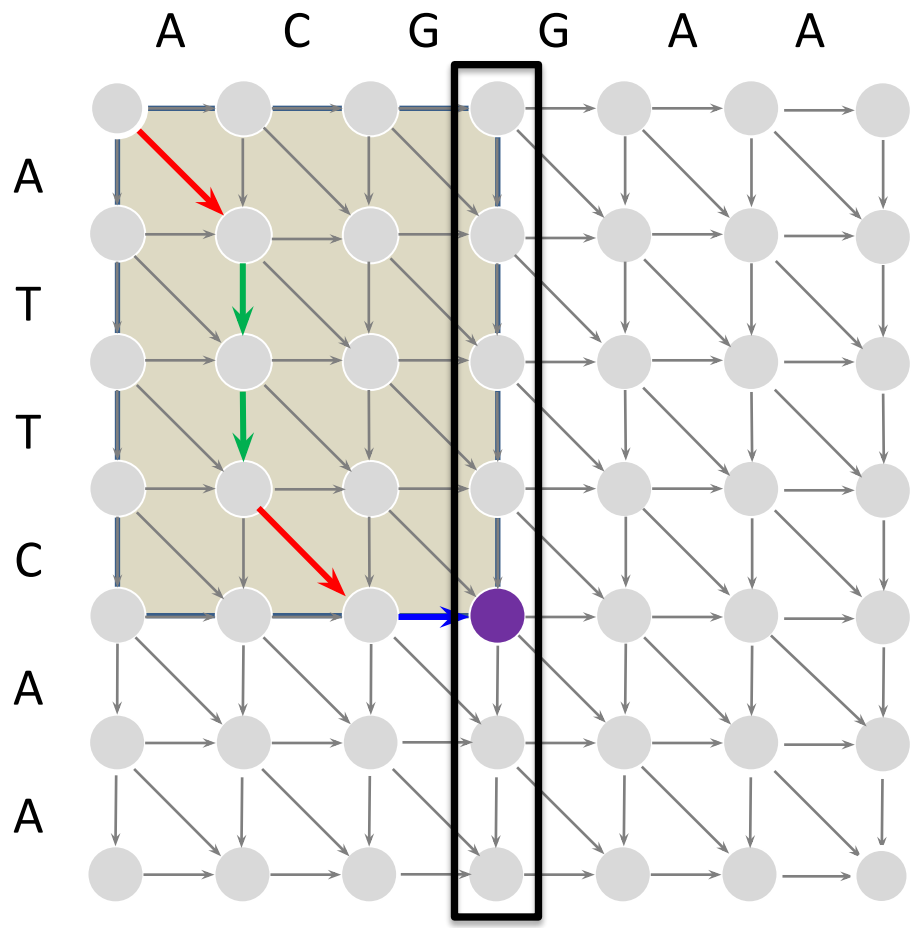


Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)



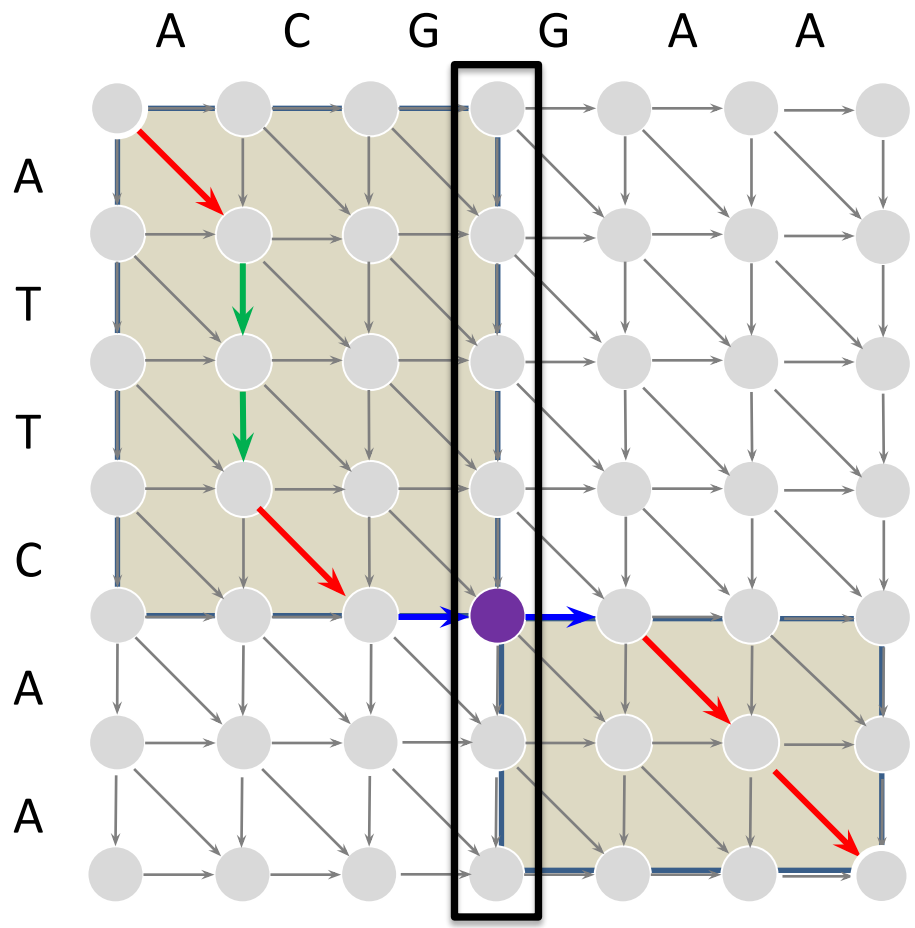
Divide and Conquer Approach to Sequence Alignment

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)

AlignmentPath(*MiddleNode, sink*)



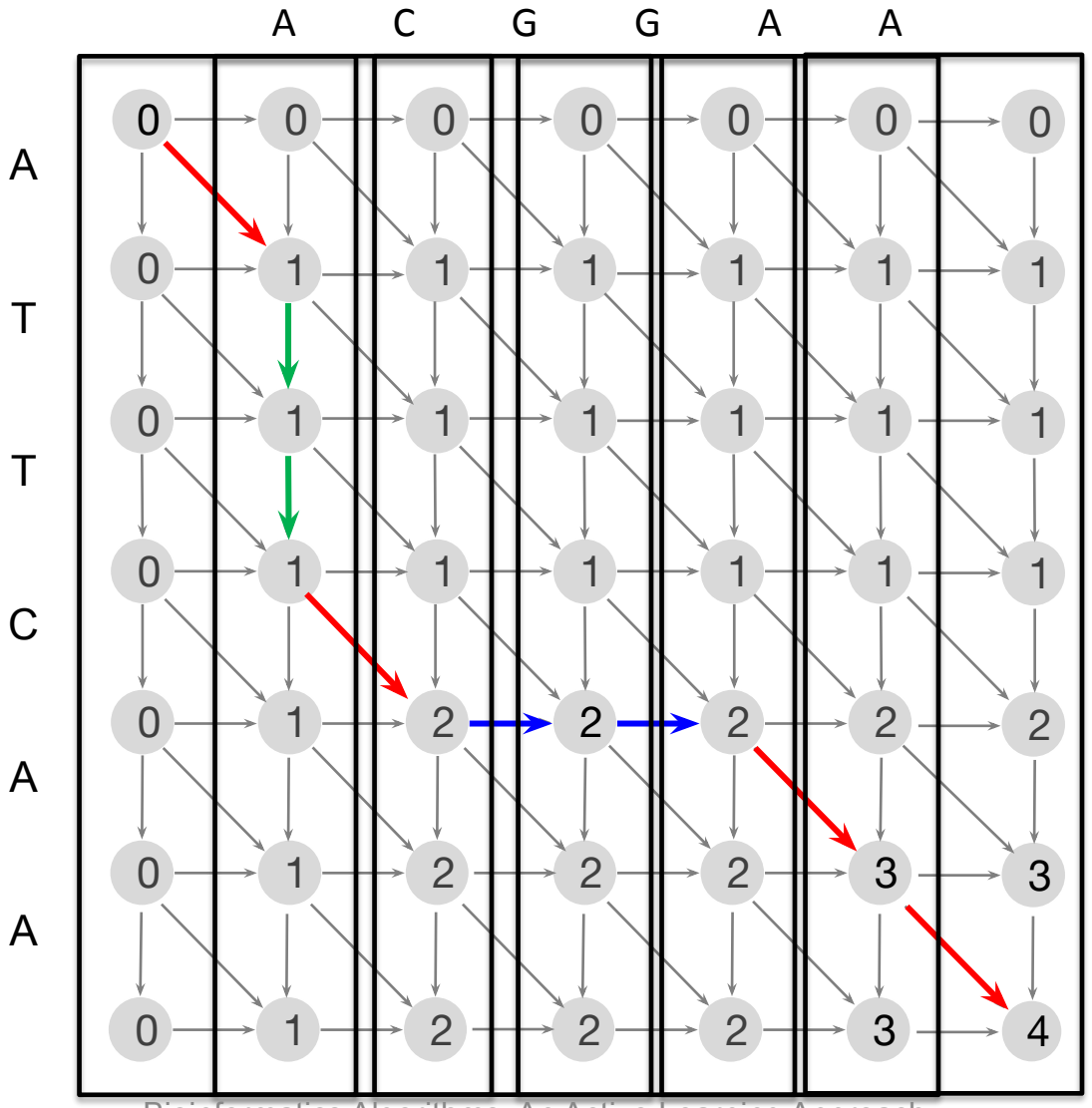
The only problem left is how to find this middle node in **linear space**!

Computing Alignment Score in Linear Space

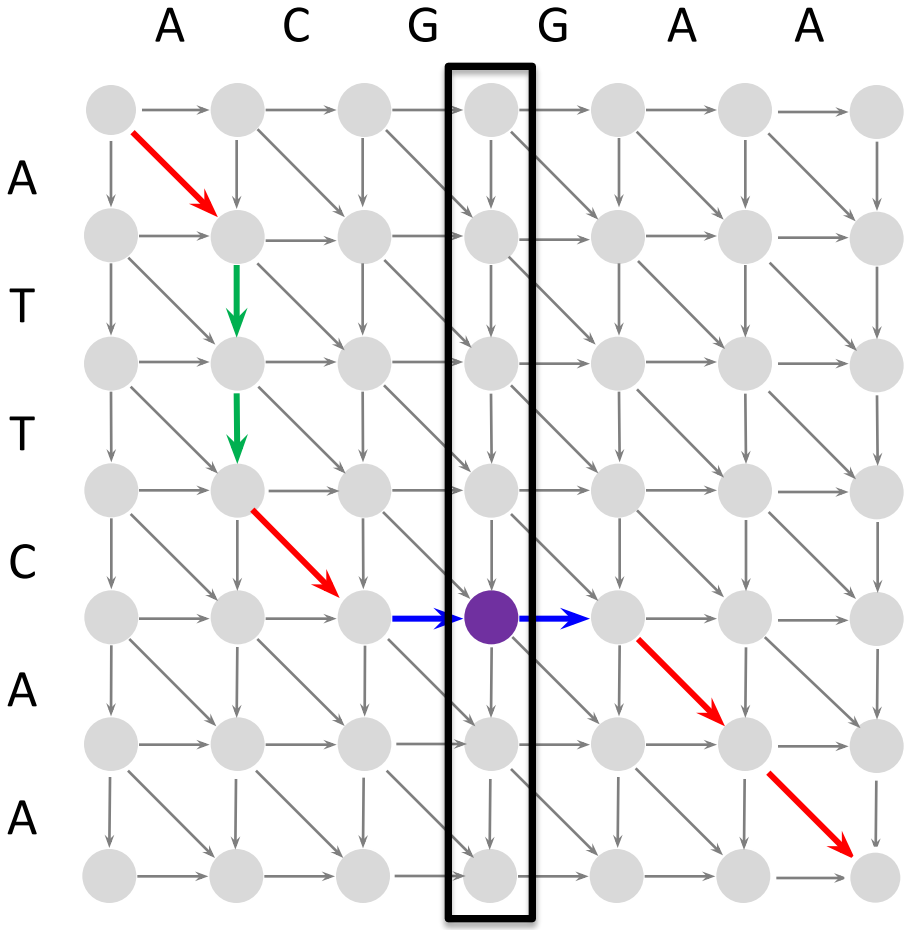
Finding the **longest path** in the alignment graph **requires** storing all backtracking pointers – O(*nm*) memory.

Finding the **length of the longest path** in the alignment graph **does not require** storing any backtracking pointers – O(*n*) memory.

Recycling the Columns in the Alignment Graph



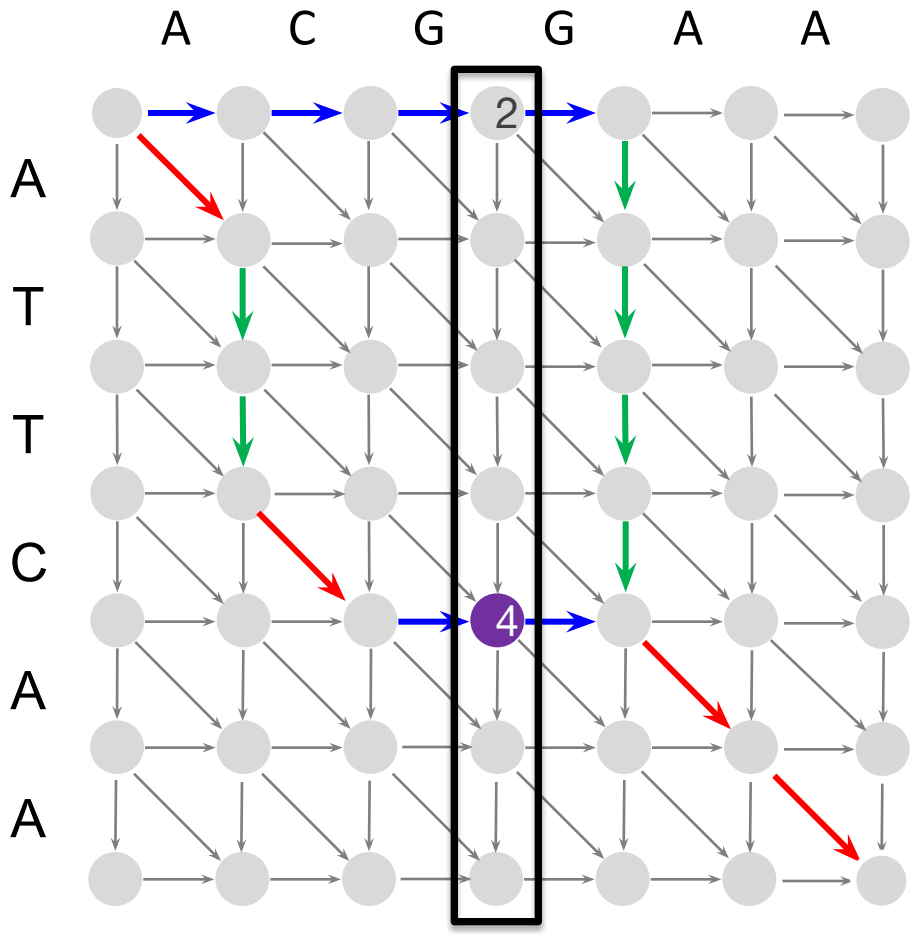
Can We Find the Middle Node without Constructing the Longest Path?



4-path that visits the node (4,middle) In the middle column

***i*-path** – a longest path among paths that visit the *i*-th node in the middle column

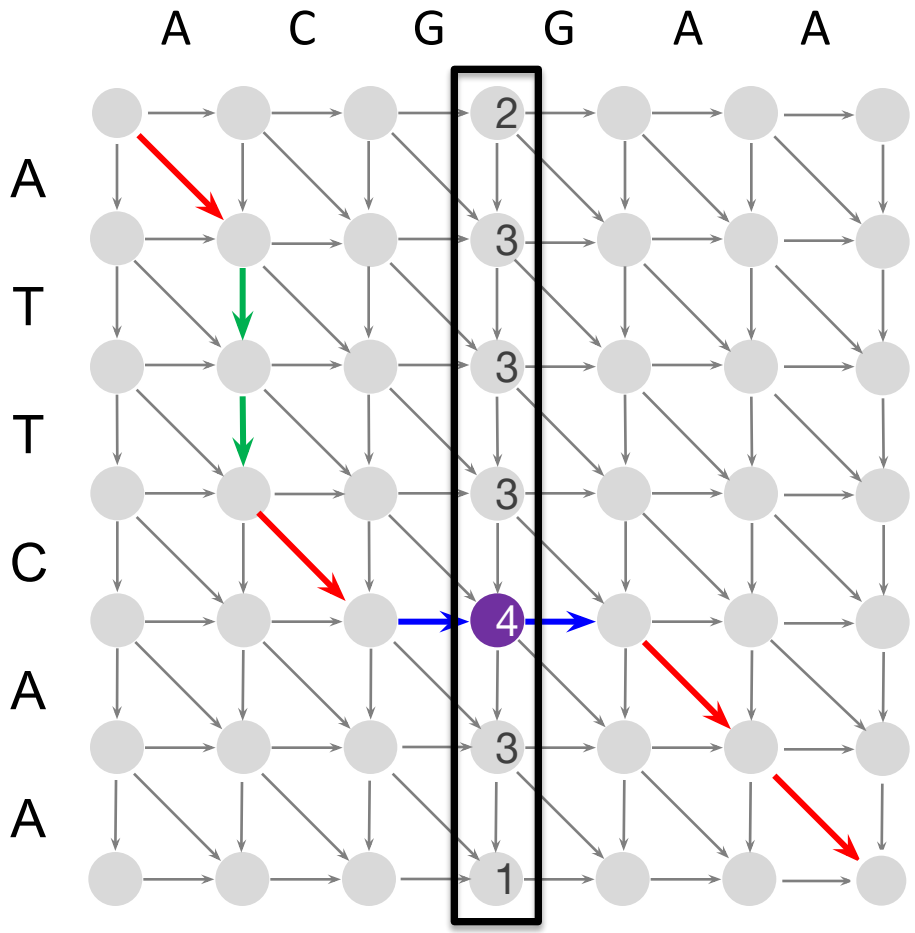
Can We Find The Lengths of All *i*-paths?



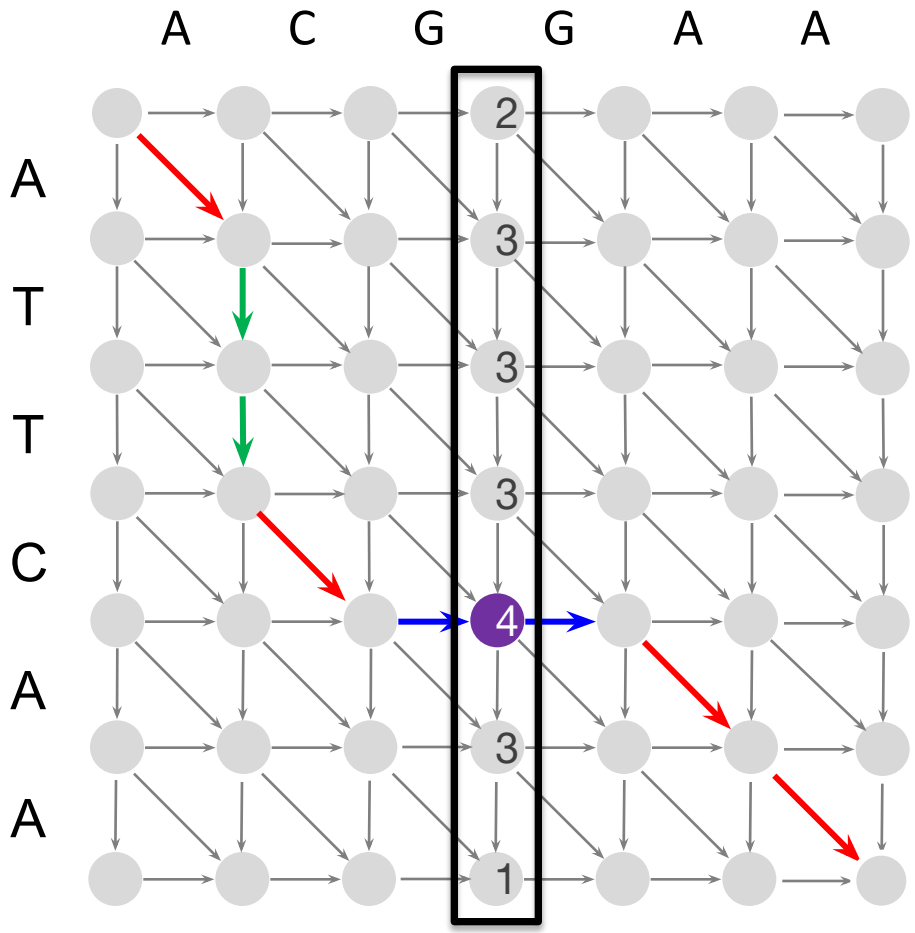
length(i):
length of an *i*-path:

length(0)=2
length(4)=4

Can We Find The Lengths of All *i*-paths?



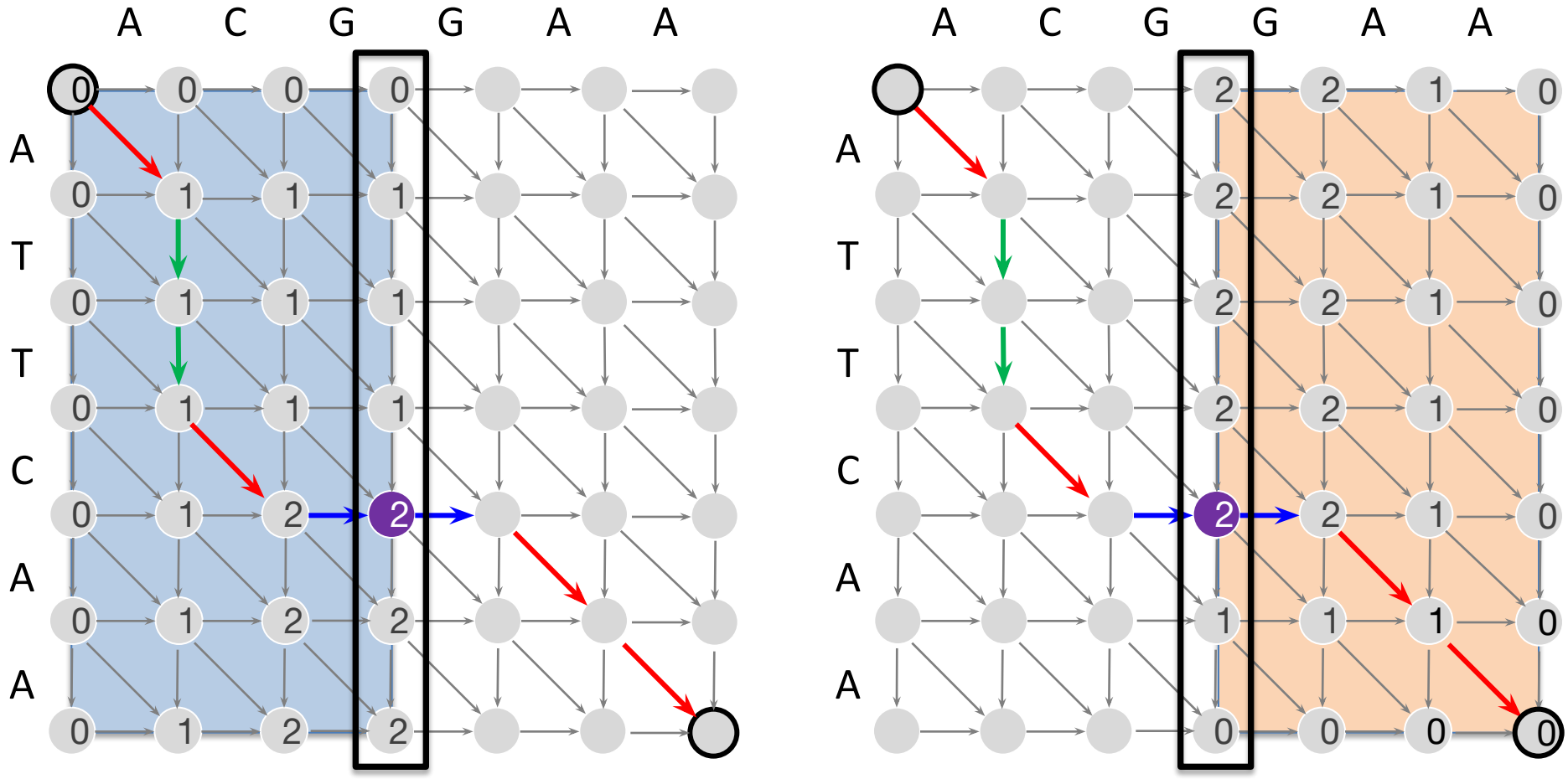
Can We Find The Lengths of *i*-paths?



length(i):
length of an *i*-path

length(i)=fromSource(i)+toSink(i)

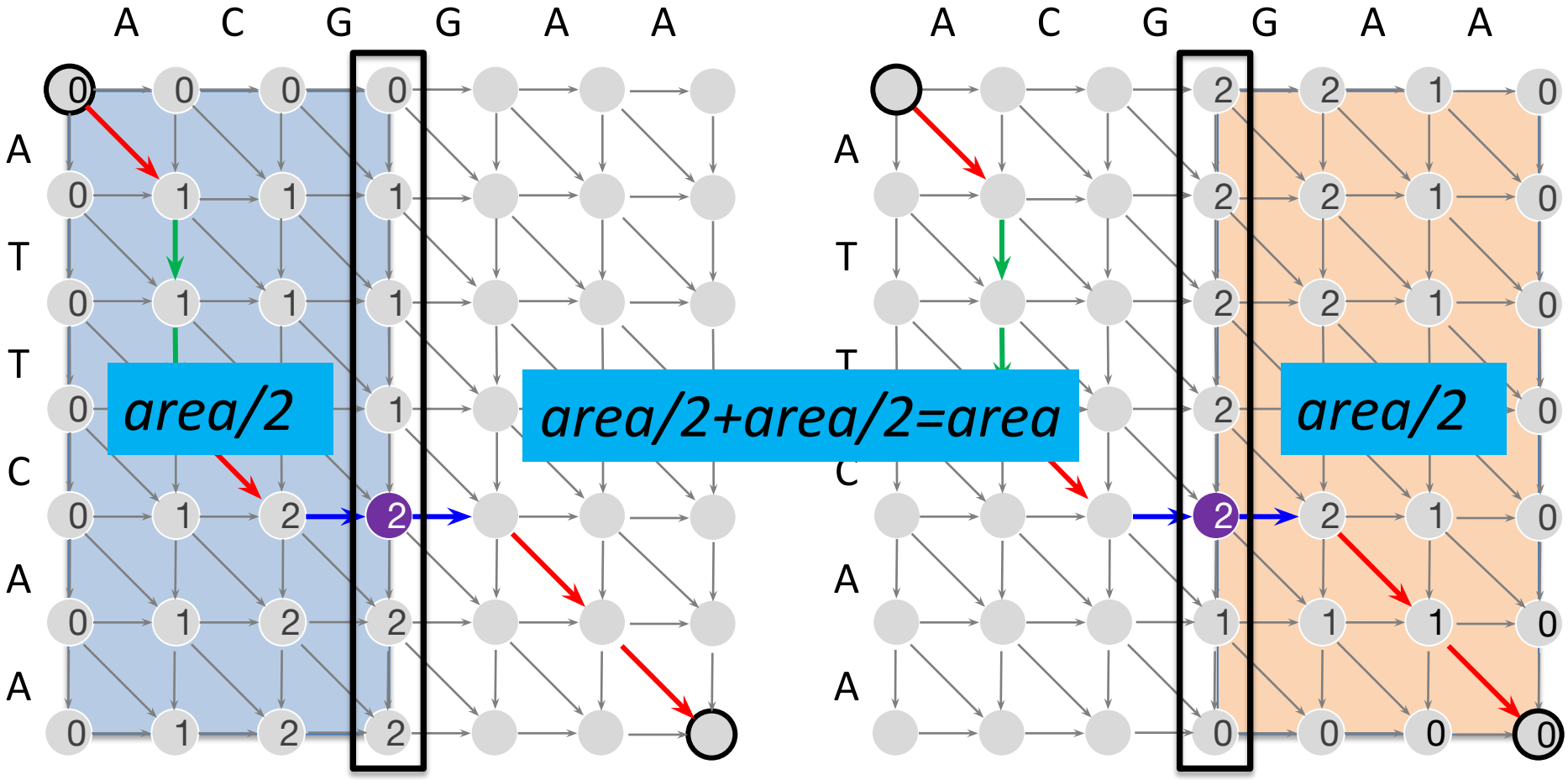
Computing *FromSource* and *toSink*



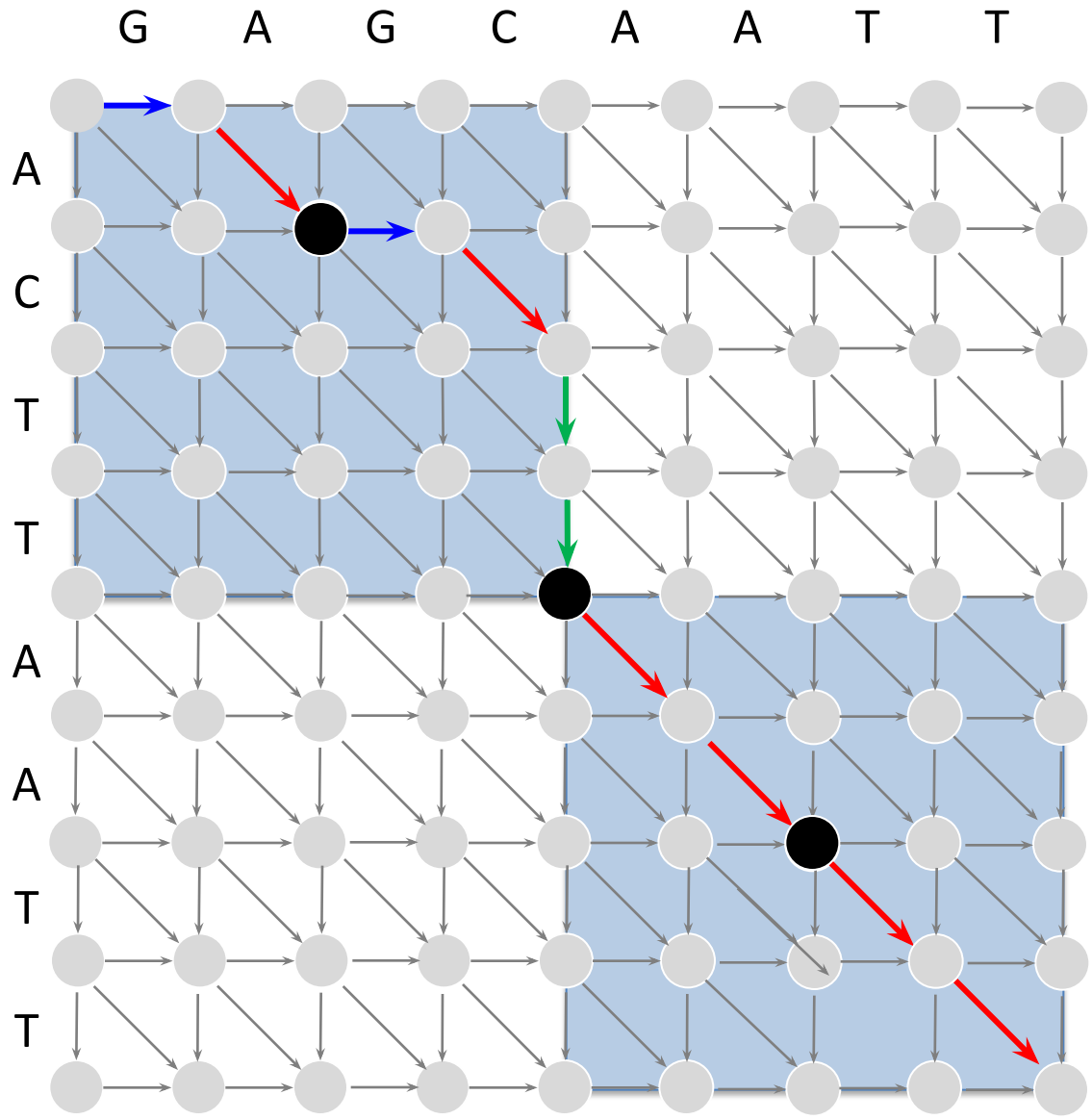
fromSource(i)

toSink(i)

How Much Time Did It Take to Find the Middle Node ?



Laughable Progress: O(*nm*) Time to Find **ONE** Node!

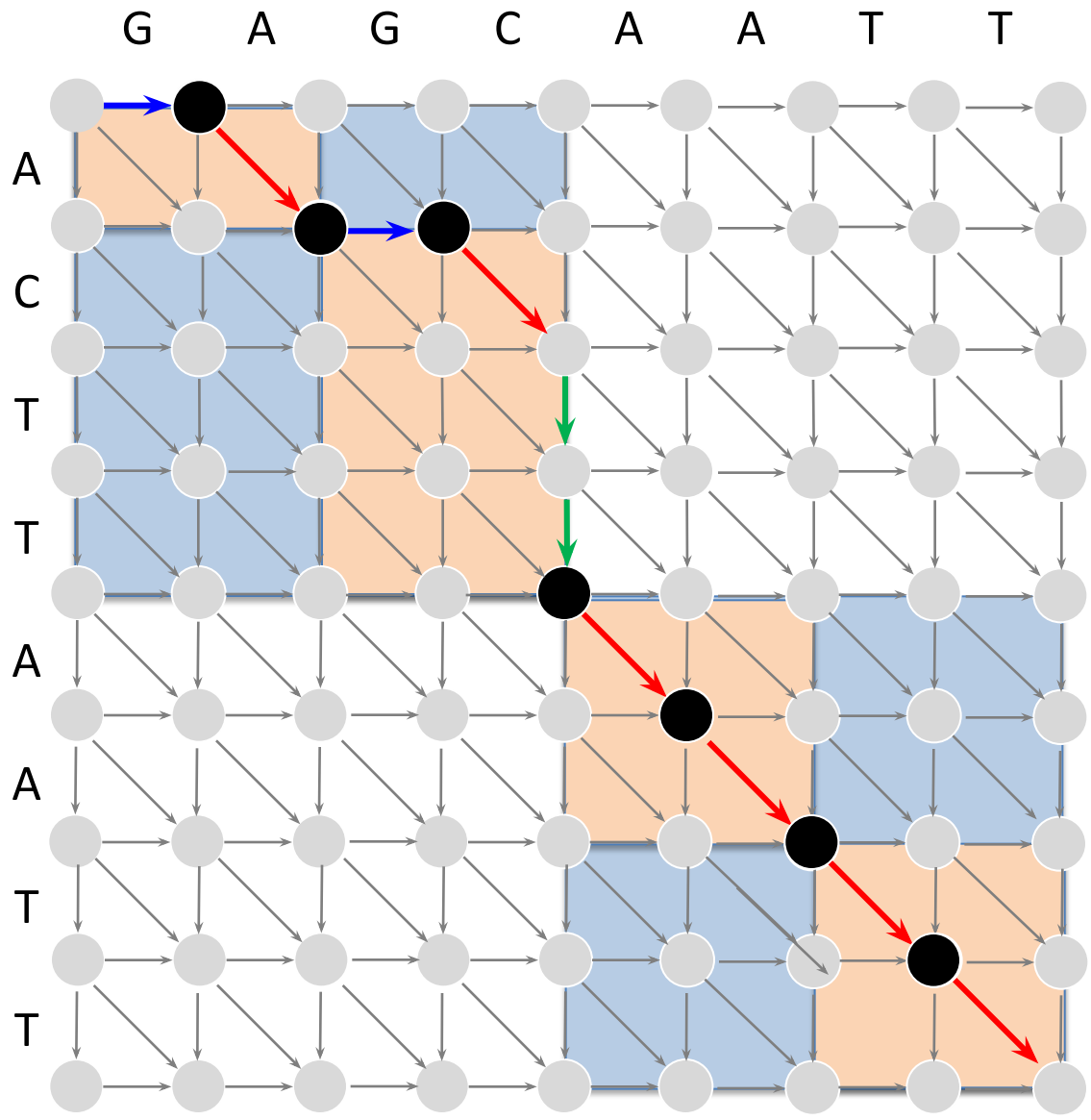


Each subproblem can be conquered in time proportional to its area:

area/4+area/4= ***area/2***

How much time would it take to conquer 2 subproblems?

Laughable Progress: O(*nm+nm/2*) Time to Find **THREE** Nodes!

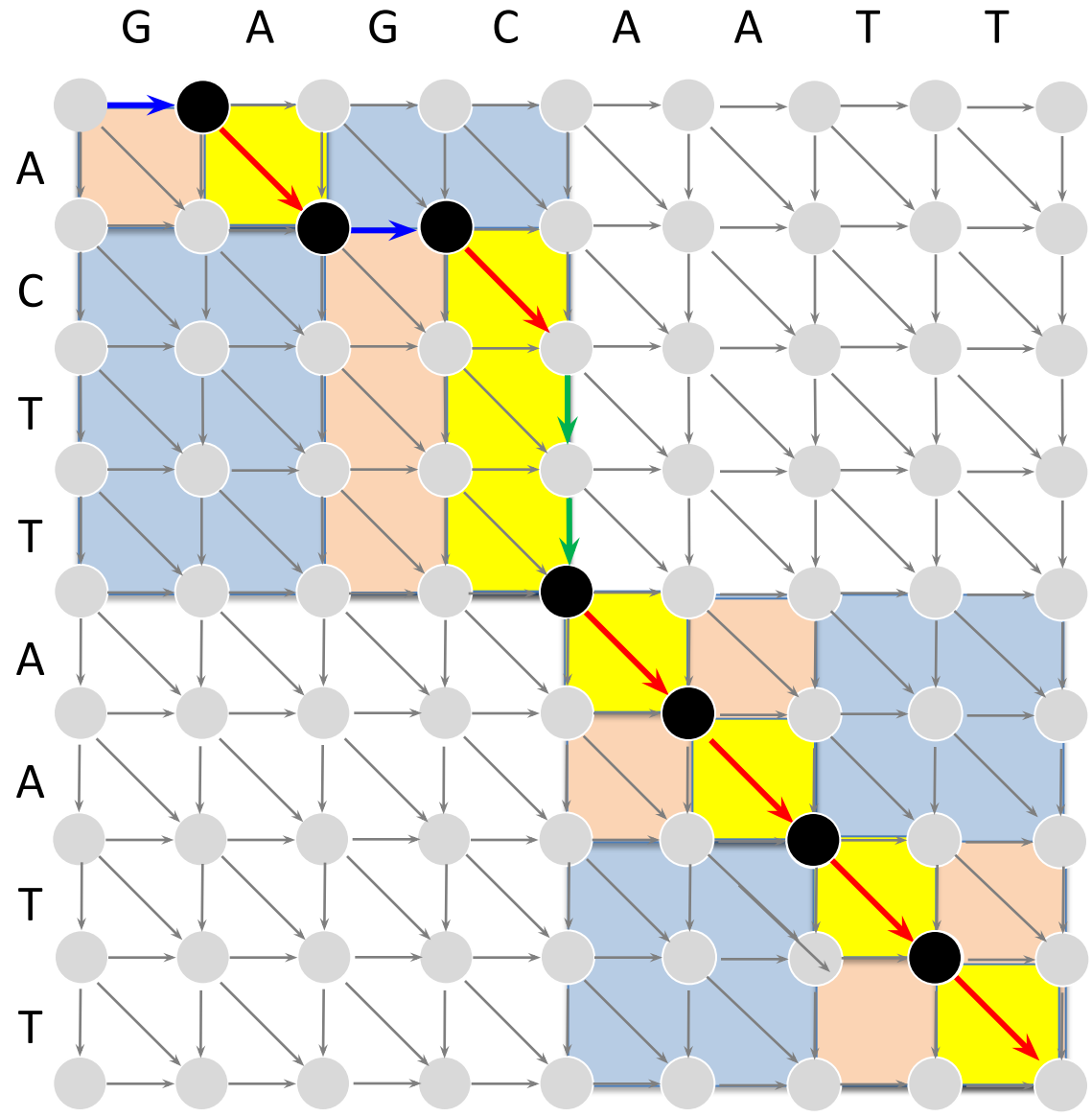


Each subproblem can be conquered in time proportional to its area:

area/8+area/8+ area/8+area/8= ***area/4***

How much time would it take to conquer 4 subproblems?

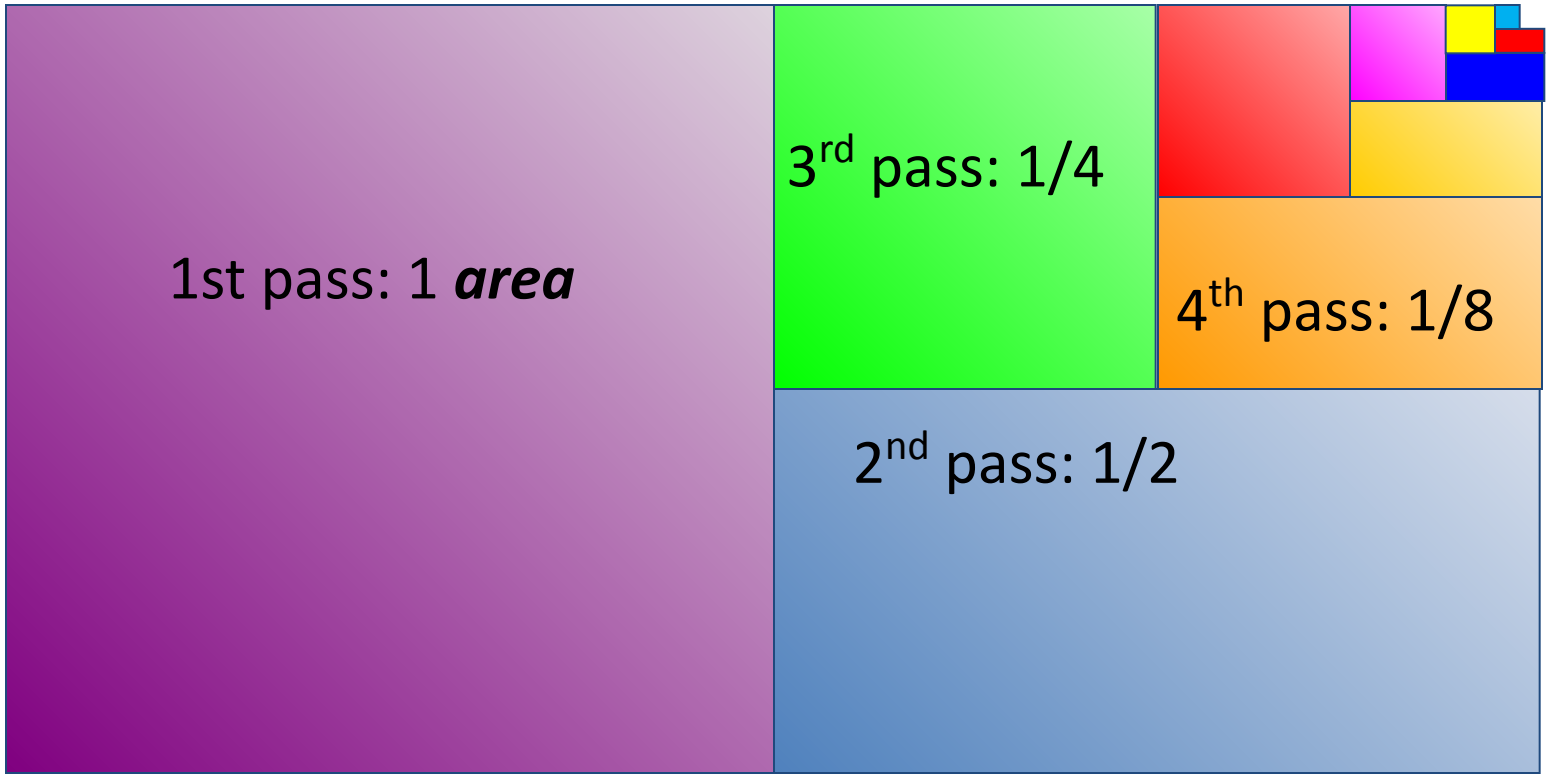
O(*nm+nm/2+nm/4*) Time to Find **NEARLY ALL** Nodes!



area+
area/2
+area/4
+area/8
+area/16
+….+
<
2·area

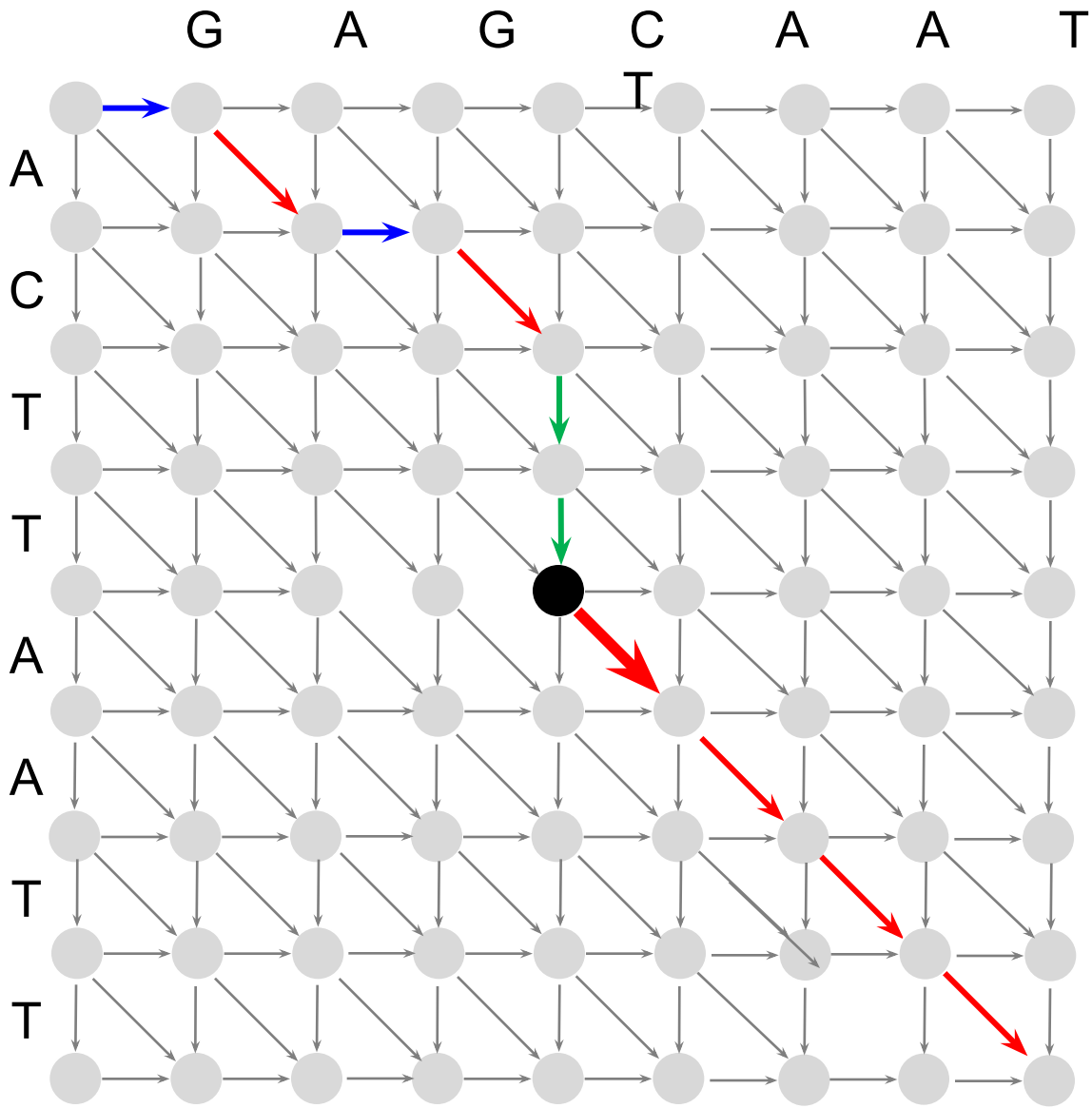
How much time would it take to conquer ALL subproblems?

Total Time:
area+area/2+area/4+area/8+area/16+…



$1 + ½ + ¼ + \ldots < 2$

The Middle Edge

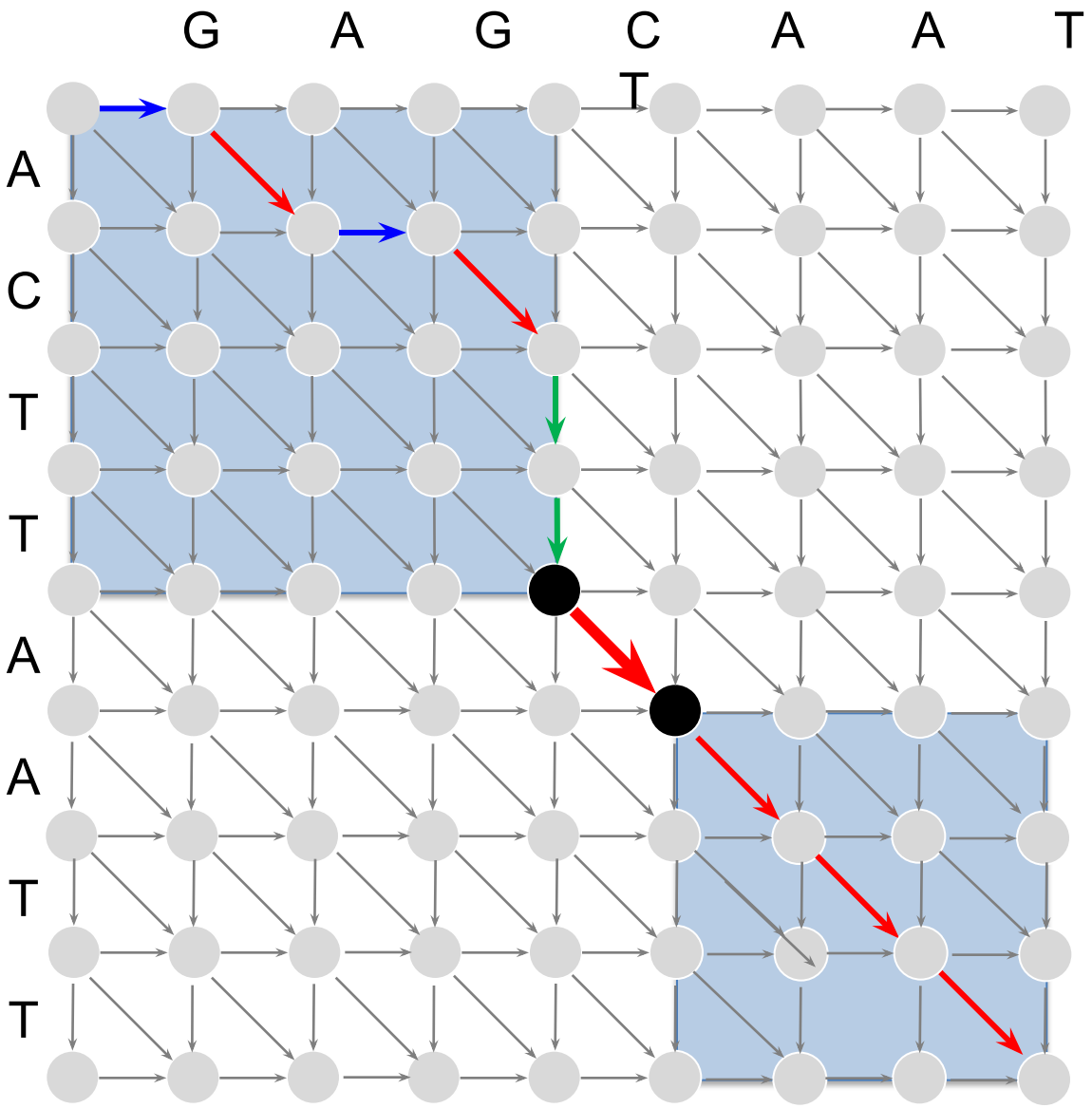


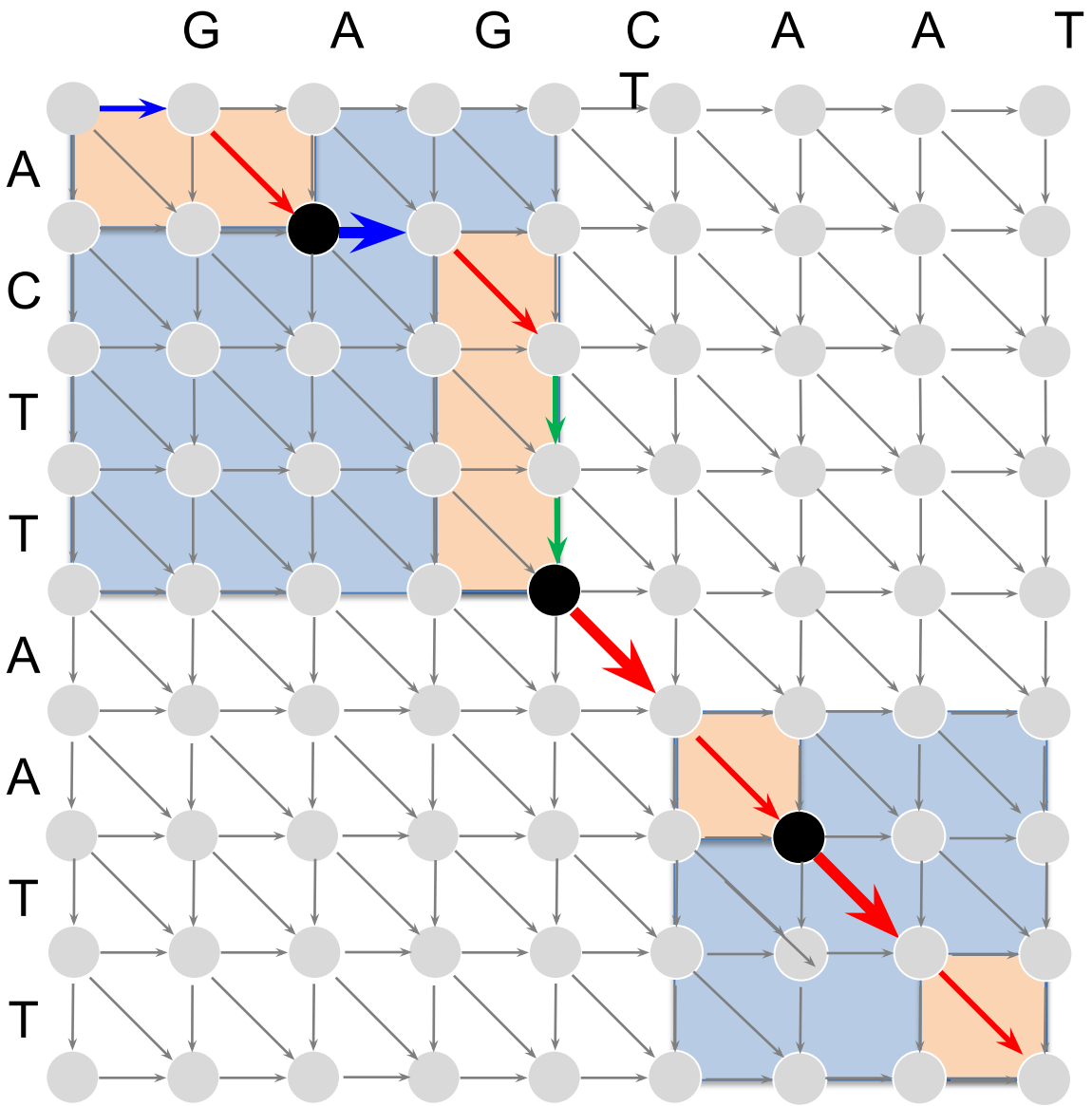
Middle Edge: an edge in an optimal alignment path starting at the middle node

The Middle Edge Problem

Middle Edge in Linear Space Problem. Find a middle edge in the alignment graph in linear space.

- **Input:** Two strings and matrix ***score***.
- **Output:** A middle edge in the alignment graph of these strings (as defined by the matrix ***score***).





Recursive **LinearSpaceAlignment**

```
LinearSpaceAlignment(top,bottom,left,right)
  if left = right
    return alignment formed by bottom-top edges "↓"
  middle ← ⌊(left+right)/2⌋
  midNode ← MiddleNode(top,bottom,left,right)
  midEdge ←  MiddleEdge(top,bottom,left,right)
  LinearSpaceAlignment(top,midNode,left,middle)
  output midEdge
  if midEdge = "→" or midEdge = "↘"
    middle  ← middle+1
  if  midEdge = "↓" or midEdge = "↘"
    midNode  ← midNode+1
  LinearSpaceAlignment(midNode,bottom,middle,right)
```

How Do We Compare Biological Sequences

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- The Manhattan Tourist Problem
- The Change Problem
- Dynamic Programming and Backtracking Pointers
- From Manhattan to an Arbitrary Directed Acyclic Graph
- From Global to Local Alignment
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- **Multiple Sequence Alignment**

From Pairwise to Multiple Alignment

- Up until now we have only tried to align two sequences.
- A faint (and statistically insignificant) similarity between two sequences becomes significant if it is present in many other sequences.
- Multiple alignments can reveal subtle similarities that pairwise alignments do not reveal.

Alignment of Three A-domains

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Alignment of Three A-domains

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Alignment of Three A-domains

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA

-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Generalizing Pairwise to Multiple Alignment

- Alignment of 2 sequences is a 2-row matrix.
- Alignment of 3 sequences is a 3-row matrix

```
A T - G C G -
A - C G T - A
A T C A C - A
```

- Our scoring function should score alignments with conserved columns higher.

Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC

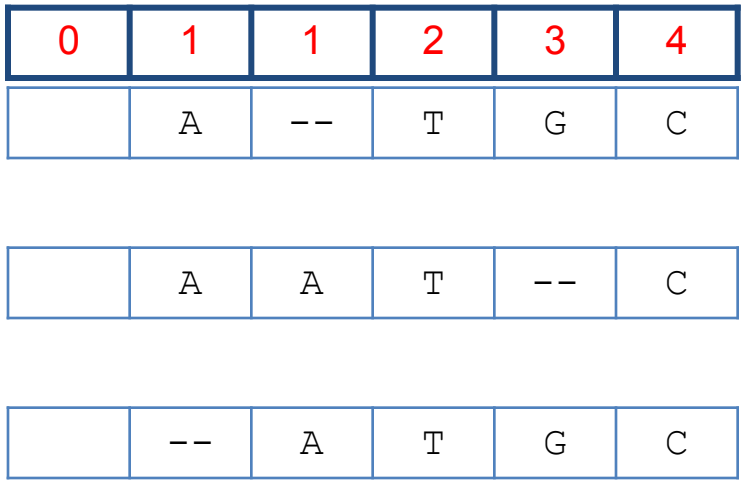
	A	--	T	G	C

	A	A	T	--	C

	--	A	T	G	C

Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC



#symbols up to a given position

Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC

0	1	1	2	3	4
	A	--	T	G	C

0	1	2	3	3	4
	A	A	T	--	C

	--	A	T	G	C

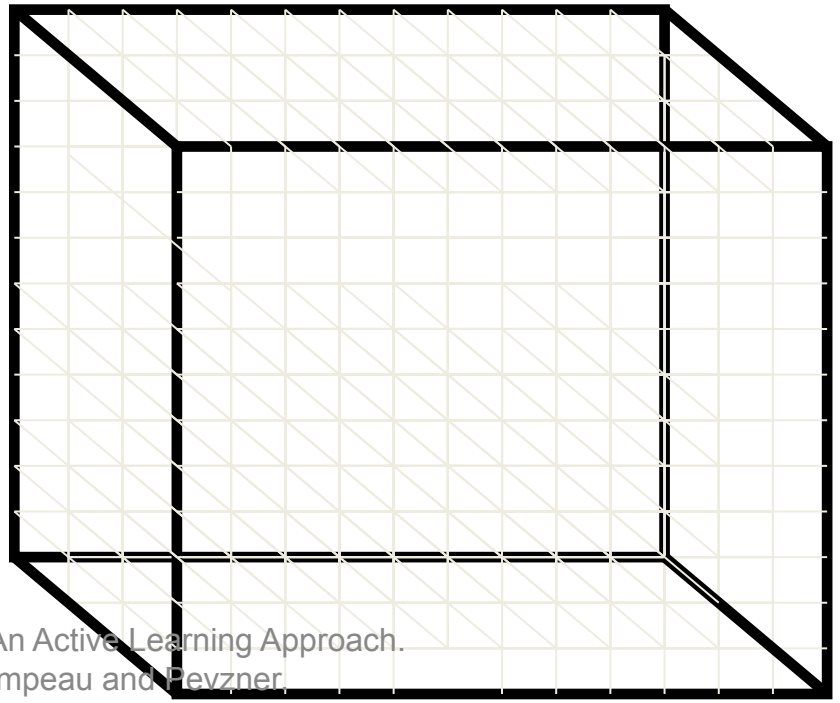
#symbols up to a given position

Alignments = Paths in 3-D

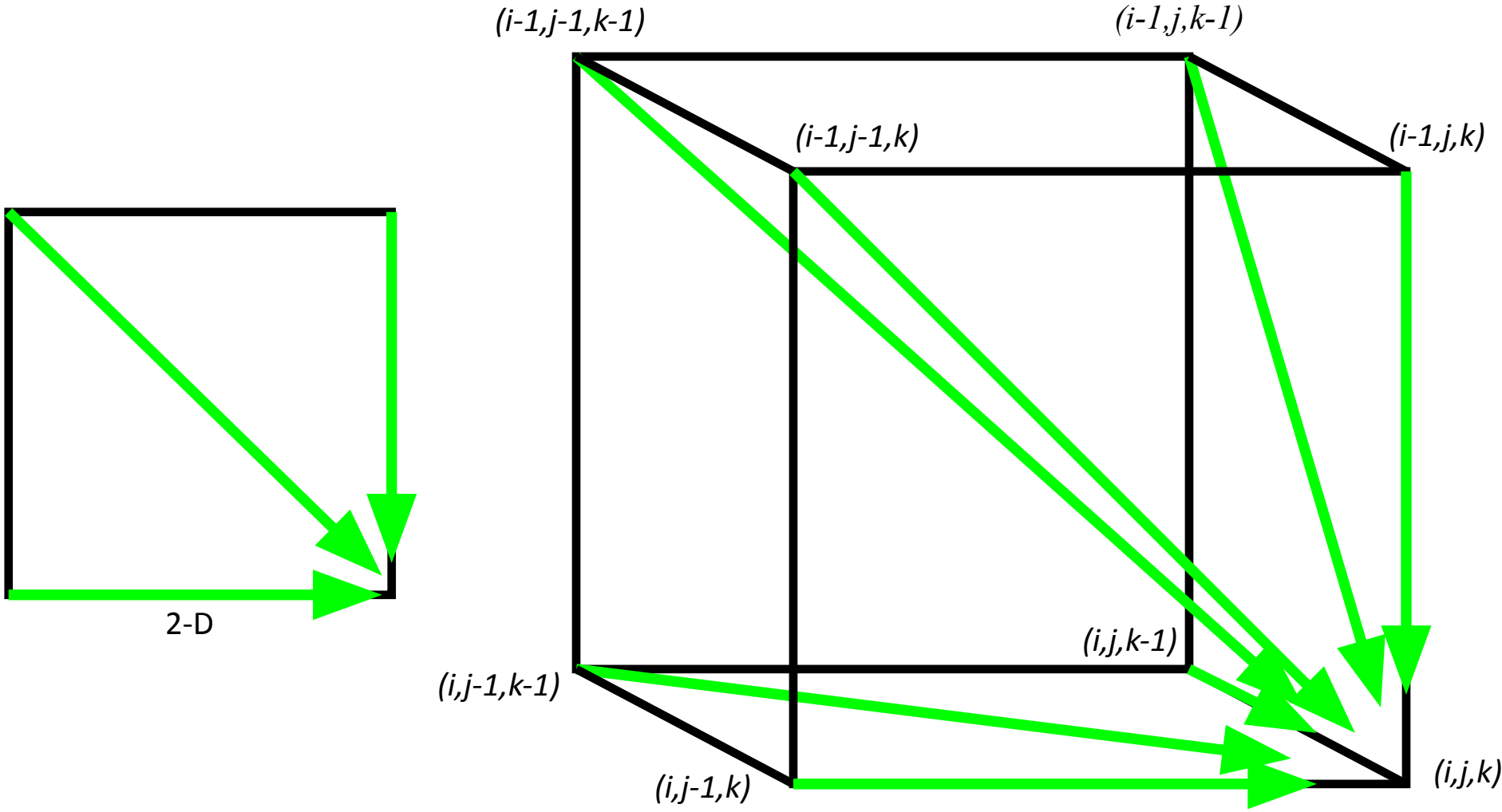
- Alignment of ATGC, AATC, and ATGC

(0,0,0)→(1,1,0)→(1,2,1) →(2,3,2) →(3,3,3) →(4,4,4)

0	1	1	2	3	4
	A	--	T	G	C
0	1	2	3	3	4
	A	A	T	--	C
0	0	1	2	3	4
	--	A	T	G	C



2-D Alignment Cell versus 3-D Alignment Cell



Multiple Alignment: Dynamic Programming

$$s_{i,j,k} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, -) \\ s_{i-1,j,k-1} + \delta(v_i, -, u_k) \\ s_{i,j-1,k-1} + \delta(-, w_j, u_k) \\ s_{i-1,j,k} + \delta(v_i, -, -) \\ s_{i,j-1,k} + \delta(-, w_j, -) \\ s_{i,j,k-1} + \delta(-, -, u_k) \end{cases}$$

- $\delta(x, y, z)$ is an entry in the 3-D scoring matrix.

Multiple Alignment: Running Time

- For 3 sequences of length n, the run time is proportional to $7n^3$
- For a k-way alignment, build a k-dimensional Manhattan graph with
  - n^k nodes
  - most nodes have $2^k - 1$ incoming edges.
  - Runtime: $O(2^k n^k)$

Multiple Alignment Induces Pairwise Alignments

Every multiple alignment induces pairwise alignments:

```
AC-GCGG-C
AC-GC-GAG
GCCGC-GAG
```

```
ACGCGG-C    AC-GCGG-C    AC-GCGAG
ACGC-GAC    GCCGC-GAG    GCCGCGAG
```

Idea: Construct Multiple from Pairwise Alignments

Given a set of **arbitrary** pairwise alignments, can we construct a multiple alignment that induces them?

```
AAAATTTT----        ----AAAATTTT        TTTTGGGG----
----TTTTGGGG        GGGGAAAA----        ----GGGGAAAA
```

Profile Representation of Multiple Alignment

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
	T	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	T	C	A	C	–	G	G
	C	A	G	–	C	T	A	T	C	G	C	–	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Aligning Sequence Against Sequence

- In the past we were aligning a **sequence against a sequence**.

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
	T	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	T	C	A	C	–	G	G
	C	A	G	–	C	T	A	T	C	G	C	–	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Aligning Sequence Against Profile

- In the past we were aligning a **sequence against a sequence**.
  - Can we align a **sequence against a profile?**

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
	T	A	G	-	C	T	A	C	C	A	-	-	-	G
	C	A	G	-	C	T	A	C	C	A	-	-	-	G
	C	A	G	-	C	T	A	T	C	A	C	-	G	G
	C	A	G	-	C	T	A	T	C	G	C	-	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Aligning Profile Against Profile

- In the past we were aligning a **sequence against a sequence.**
  - Can we align a **sequence against a profile?**
  - Can we align a **profile against a profile?**

```
      -  A  G  G  C  T  A  T  C  A  C  C  T  G
      T  A  G  -  C  T  A  C  C  A  -  -  -  G
      C  A  G  -  C  T  A  C  C  A  -  -  -  G
      C  A  G  -  C  T  A  T  C  A  C  -  G  G
      C  A  G  -  C  T  A  T  C  G  C  -  G  G

A    0  1  0  0  0  0  1  0  0 .8  0  0  0  0
C   .6  0  0  0  1  0  0 .4  1  0 .6 .2  0  0
G    0  0  1 .2  0  0  0  0  0 .2  0  0 .4  1
T   .2  0  0  0  0  1  0 .6  0  0  0  0 .2  0
-   .2  0  0 .8  0  0  0  0  0  0 .4 .8 .4  0
```

Multiple Alignment: Greedy Approach

- Choose the most similar sequences and combine them into a profile, thereby reducing alignment of *k* sequences to an alignment of of *k* – 2 sequences and 1 profile.
- Iterate

Greedy Approach: Example

- Sequences: GATTCA, GTCTGA, GATATT, GTCAGC.

- 6 pairwise alignments (premium for match +1, penalties for indels and mismatches -1)

```
s2  GTCTGA
s4  GTCAGC  (score = 2)

s1  GAT-TCA
s2  G-TCTGA (score = 1)

s1  GAT-TCA
s3  GATAT-T (score = 1)

s1  GATTCA--
s4  G-T-CAGC (score = 0)

s2  G-TCTGA
s3  GATAT-T  (score = -1)

s3  GAT-ATT
s4  G-TCAGC  (score = -1)
```

Greedy Approach: Example

- Since s_2 and s_4 are closest, we consolidate them into a profile:

$$\left.\begin{matrix} s2\text{GTCTGA} \\ s4\text{GTCAGC} \end{matrix}\right\} s_{2,4} = \text{GTCt/aGa/cA}$$

- New set of 3 sequences to align:

s_1 GATTCA
s_3 GATATT
$s_{2,4}$ GTCt/aGa/c