AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

22BIO211: Intelligence of Biological Systems - 2

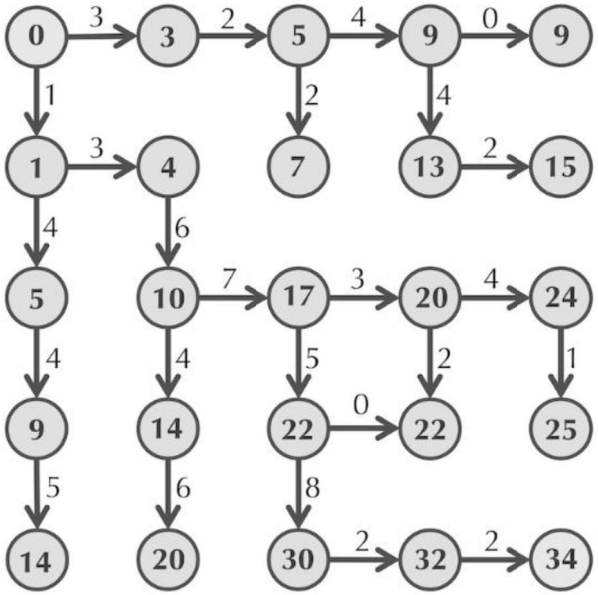
BACKTRACKING IN ALIGNMENT GRAPHS

Dr. Manjusha Nair M
Amrita School of Computing, Amritapuri

Email : manjushanair@am.amrita.edu
Contact No: 9447745519

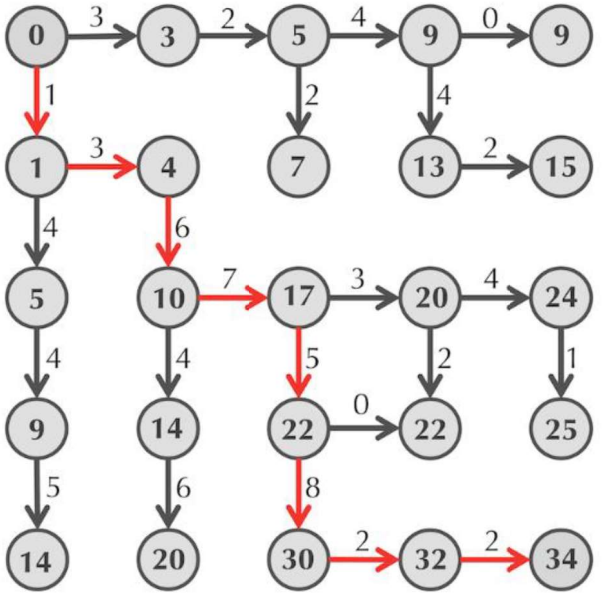
BackTracking

- Each edge selected by ManhattanTourist.
- To form a longest path, we simply need to find a path from source to sink formed by highlighted edges (more than one such path may exist).
- However, if we were to walk from source to sink along the highlighted edges, we might reach a dead end, such as the node (1, 2).



Backtracking

- In contrast, every path from the sink will bring us back to the source if we backtrack in the direction opposite to each highlighted edge.
- Following edges from the sink backwards to the source produces the red path in the DAG , which highlights a longest path from the source to sink.
- We can use this backtracking idea to construct an LCS of strings v and w.



Backtracking - Algorithm

- If we assign a weight of 1 to the edges in AlignmentGraph(v, w) corresponding to matches and assign a weight of 0 to all other edges, then $s_{|v|, |w|}$ gives the length of an LCS.
- The following algorithm maintains a record of which edge was used to compute each value $s_{i, j}$ by utilizing backtracking pointers, which take one of the three values $\downarrow$, $\rightarrow$, or $\searrow$.
- Backtracking pointers are stored in a matrix ***Backtrack***.

Backtracking - Algorithm

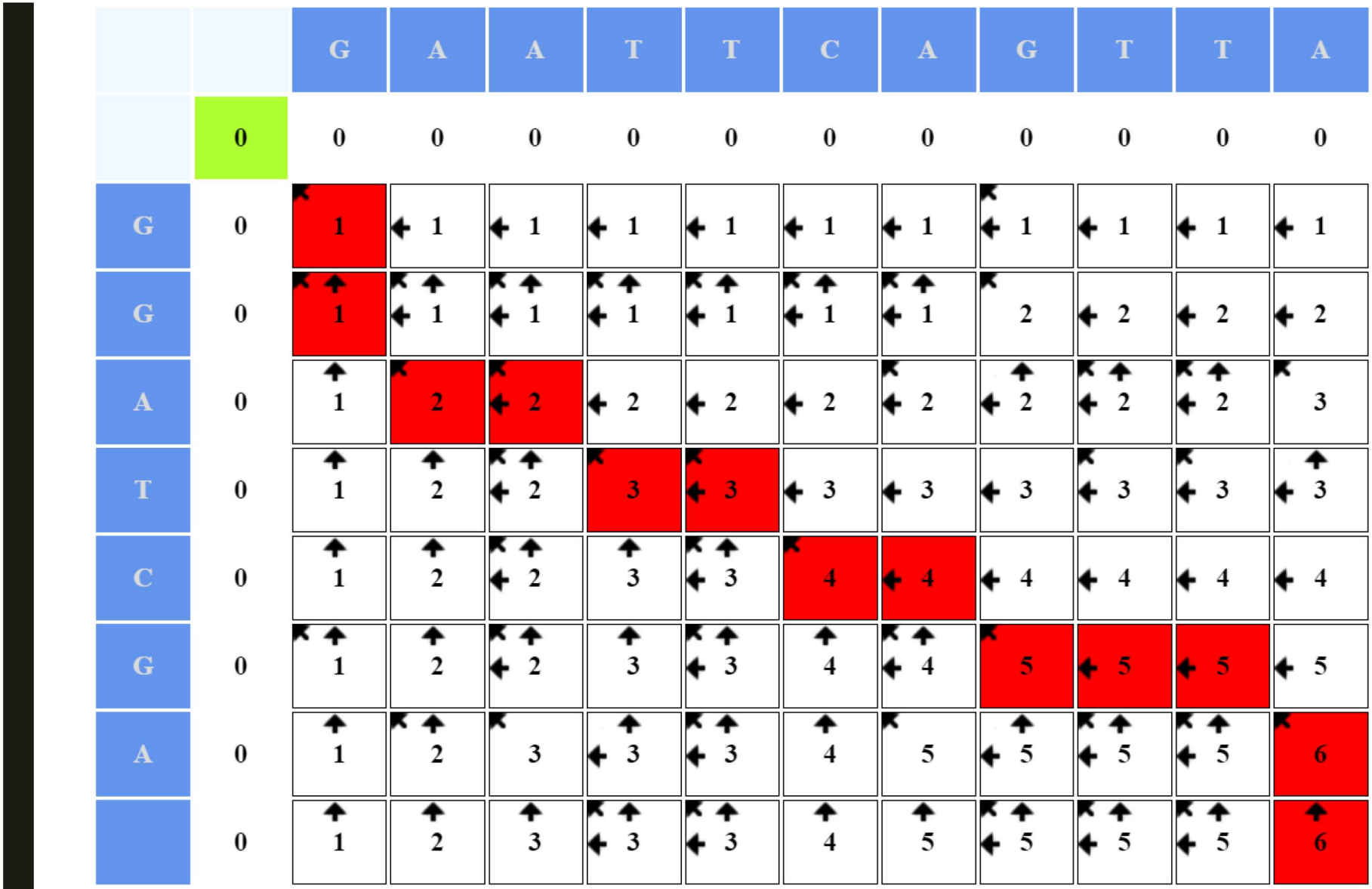
```
LCSBackTrack(v, w)
  for i ← 0 to |v|
    s_{i, 0} ← 0
  for j ← 0 to |w|
    s_{0, j} ← 0
  for i ← 1 to |v|
    for j ← 1 to |w|
      match ← 0
      if v_{i-1} = w_{j-1}
        match ← 1
      s_{i, j} ← max{s_{i-1, j} , s_{i,j-1} , s_{i-1, j-1} + match }
      if s_{i,j} = s_{i-1,j}
        Backtrack_{i, j} ← "↓"
      else if s_{i, j} = s_{i, j-1}
        Backtrack_{i, j} ← "→"
      else if s_{i, j} = s_{i-1, j-1} + match
        Backtrack_{i, j} ← "↘"
  return Backtrack
```

Longest Path in a DAG Problem

- We now need to find a path from the source to the sink formed by the highlighted edges.
- We can use this backtracking idea to construct an LCS of strings $\boldsymbol{v}$ and $\boldsymbol{w}$.
- The algorithm in the next slide solves the Longest Common Subsequence Problem by using the information in *Backtrack*.
- ***OUTPUTLCS****(Backtrack, v, i, j)* outputs an LCS between the *i-prefix* of v and the *j-prefix* of w.
- The initial invocation that outputs an LCS of v and w is ***OUTPUTLCS****(Backtrack, v, |v|, |w|)*.

Longest Path in a DAG Problem

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


G A A T T C A G T T A
0 0 0 0 0 0 0 0 0 0 0 0
G 0 1 1 1 1 1 1 1 1 1 1 1
G 0 1 1 1 1 1 1 1 2 2 2 2
A 0 1 2 2 2 2 2 2 2 2 2 3
T 0 1 2 2 3 3 3 3 3 3 3 3
C 0 1 2 2 3 3 4 4 4 4 4 4
G 0 1 2 2 3 3 4 4 5 5 5 5
A 0 1 2 3 3 3 4 5 5 5 5 6
0 1 2 3 3 3 4 5 5 5 5 6

Sequence alignment Example -Backtracking

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	1	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	4	4	4	4	4	4
G	0	1	2	2	3	3	4	4	5	5	5	5
A	0	1	2	3	3	3	4	5	5	5	5	6

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	
G	0	1	1	1	1	1	1	1	1	1	1	
G	0	1	1	1	1	1	1	1	2	2	2	
A	0	1	2	2	2	2	2	2	2	2	2	
T	0	1	2	2	3	3	3	3	3	3	3	
C	0	1	2	2	3	3	4	4	4	4	4	
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

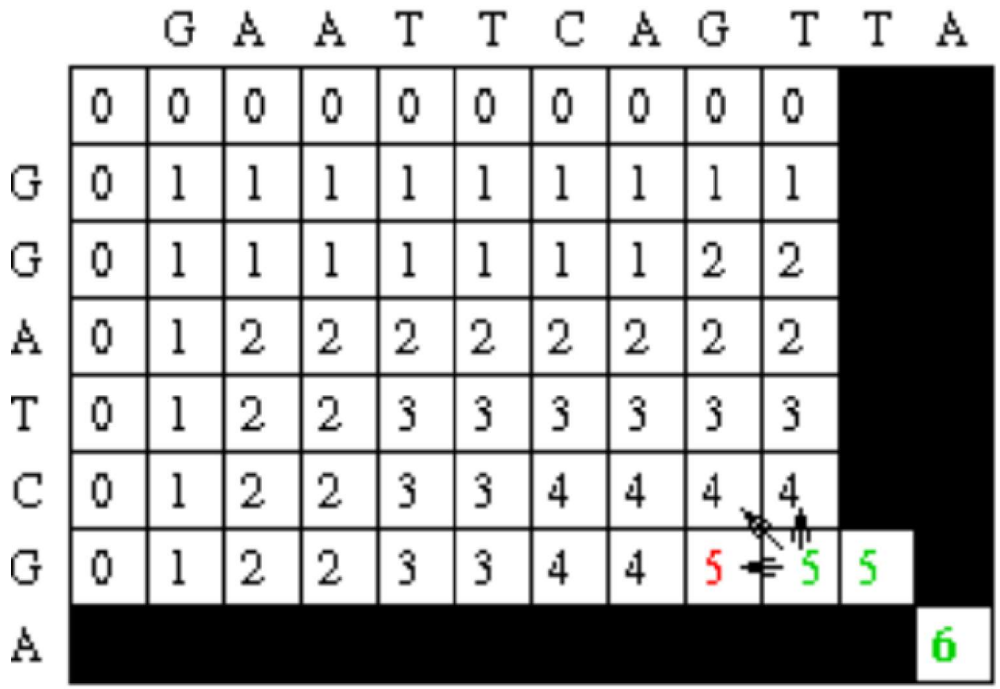
Alignment:

(Seq #1) A

|

(Seq #2) A

Sequence alignment Example -Backtracking



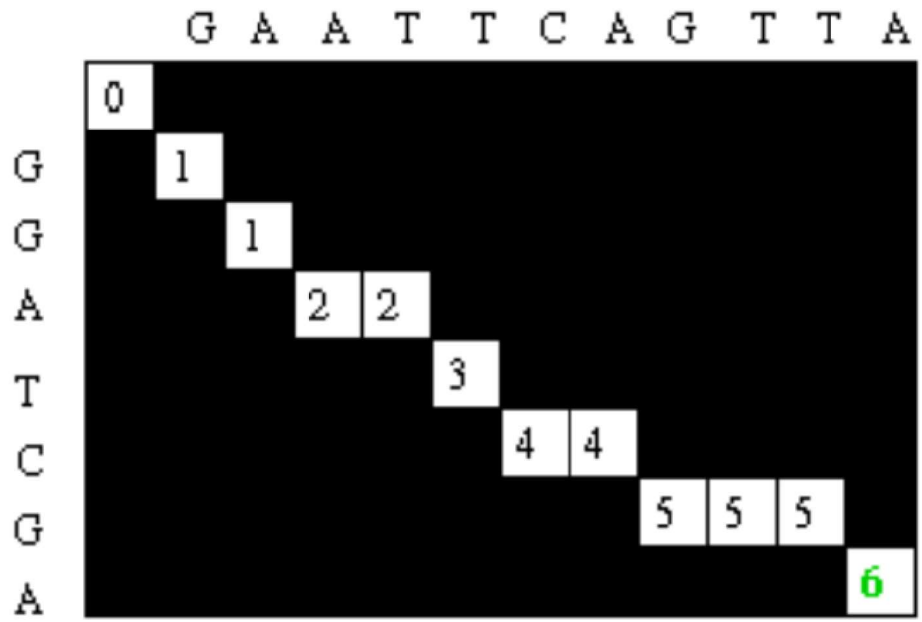
		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0		
G	0	1	1	1	1	1	1	1	1	1		
G	0	1	1	1	1	1	1	1	2	2		
A	0	1	2	2	2	2	2	2	2	2		
T	0	1	2	2	3	3	3	3	3	3		
C	0	1	2	2	3	3	4	4	4	4		
G	0	1	2	2	3	3	4	4	5	5	5	
A											6	

Alignment:

```
(Seq #1)   T A
             |
(Seq #2)   - A
```

Sequence alignment Example -Backtracking

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0		
G	0	1	1	1	1	1	1	1	1	1		
G	0	1	1	1	1	1	1	1	2	2		
A	0	1	2	2	2	2	2	2	2	2		
T	0	1	2	2	3	3	3	3	3	3		
C	0	1	2	2	3	3	4	4	4	4		
G	0	1	2	2	3	3	4	4	5	5	5	
A												6



Alignment:

```
(Seq #1) G A A T T C A G T T A
         |   |   | | |   |     |
(Seq #2) G G A - T C - G - - A
```

Scoring Alignment

- To generalize the alignment scoring model
  - *we award +1 for matches,*
  - *also penalize mismatches by some positive constant μ (the mismatch penalty)*
  - *and indels by some positive constant σ (the indel penalty).*
- As a result, the score of an alignment is equal to the following expression:

$$S = \#\text{ matches} - \mu \cdot \#\text{ mismatches} - \sigma \cdot \#\text{ indels}$$

Scoring Alignment

- For example, with the parameters μ = 1 and σ = 2, the following alignment will be assigned a score of -4.

A	T	–	G	T	T	A	T	A
A	T	C	G	T	–	C	–	C
+1	+1	-2	+1	+1	-2	-1	-2	-1

- s = 4 - (1 x 2) - (2 x 3)= 4 - 2 - 6 = -4
- Biologists have further refined this cost function to allow for the fact that
  - *some mutations may be more likely than others*
    - which calls for mismatches and indel penalties that differ depending on the specific symbols involved.

Scoring Alignment

- We will extend the k-letter alphabet to include the space symbol and then construct a $(k + 1) \times (k + 1)$ scoring matrix holding the score of aligning every pair of symbols.
- The scoring matrix for comparing DNA sequences ($k = 4$) when all mismatches are penalized by μ and all indels are penalized by σ is shown below.

	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 matrices for protein sequence comparison

- Scoring matrices for DNA sequence comparison are usually defined only by the parameters μ and σ
- Scoring matrices for protein sequence comparison weight different mutations differently and become quite involved.
- One example- the PAM_{250} scoring matrix (Here, the indel penalty has been set to 8)

PAM_{250} scoring matrix

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y	–
A	2	-2	0	0	-3	1	-1	-1	-1	-2	-1	0	1	0	-2	1	1	0	-6	-3	-8
C	-2	12	-5	-5	-4	-3	-3	-2	-5	-6	-5	-4	-3	-5	-4	0	-2	-2	-8	0	-8
D	0	-5	4	3	-6	1	1	-2	0	-4	-3	2	-1	2	-1	0	0	-2	-7	-4	-8
E	0	-5	3	4	-5	0	1	-2	0	-3	-2	1	-1	2	-1	0	0	-2	-7	-4	-8
F	-3	-4	-6	-5	9	-5	-2	1	-5	2	0	-3	-5	-5	-4	-3	-3	-1	0	7	-8
G	1	-3	1	0	-5	5	-2	-3	-2	-4	-3	0	0	-1	-3	1	0	-1	-7	-5	-8
H	-1	-3	1	1	-2	-2	6	-2	0	-2	-2	2	0	3	2	-1	-1	-2	-3	0	-8
I	-1	-2	-2	-2	1	-3	-2	5	-2	2	2	-2	-2	-2	-2	-1	0	4	-5	-1	-8
K	-1	-5	0	0	-5	-2	0	-2	5	-3	0	1	-1	1	3	0	0	-2	-3	-4	-8
L	-2	-6	-4	-3	2	-4	-2	2	-3	6	4	-3	-3	-2	-3	-3	-2	2	-2	-1	-8
M	-1	-5	-3	-2	0	-3	-2	2	0	4	6	-2	-2	-1	0	-2	-1	2	-4	-2	-8
N	0	-4	2	1	-3	0	2	-2	1	-3	-2	2	0	1	0	1	0	-2	-4	-2	-8
P	1	-3	-1	-1	-5	0	0	-2	-1	-3	-2	0	6	0	0	1	0	-1	-6	-5	-8
Q	0	-5	2	2	-5	-1	3	-2	1	-2	-1	1	0	4	1	-1	-1	-2	-5	-4	-8
R	-2	-4	-1	-1	-4	-3	2	-2	3	-3	0	0	0	1	6	0	-1	-2	2	-4	-8
S	1	0	0	0	-3	1	-1	-1	0	-3	-2	1	1	-1	0	2	1	-1	-2	-3	-8
T	1	-2	0	0	-3	0	-1	0	0	-2	-1	0	0	-1	-1	1	3	0	-5	-3	-8
V	0	-2	-2	-2	-1	-1	-2	4	-2	2	2	-2	-1	-2	-2	-1	0	4	-6	-2	-8
W	-6	-8	-7	-7	0	-7	-3	-5	-3	-2	-4	-4	-6	-5	2	-2	-5	-6	17	0	-8
Y	-3	0	-4	-4	7	-5	0	-1	-4	-1	-2	-2	-5	-4	-4	-3	-3	-2	0	10	-8
–	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	

Summary

- BackTracking
  - *Algorithm*
- Longest Path in a DAG Problem
- Scoring Alignment
- Scoring matrices for protein sequence comparison
  - *PAM_{250} scoring matrix*