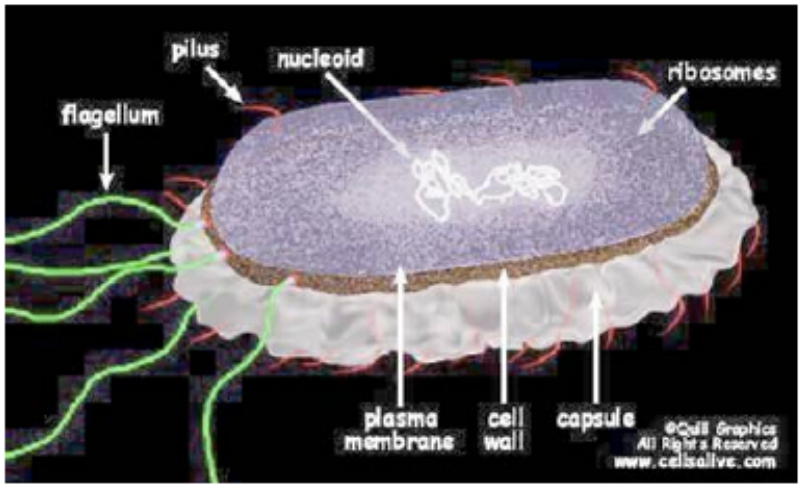
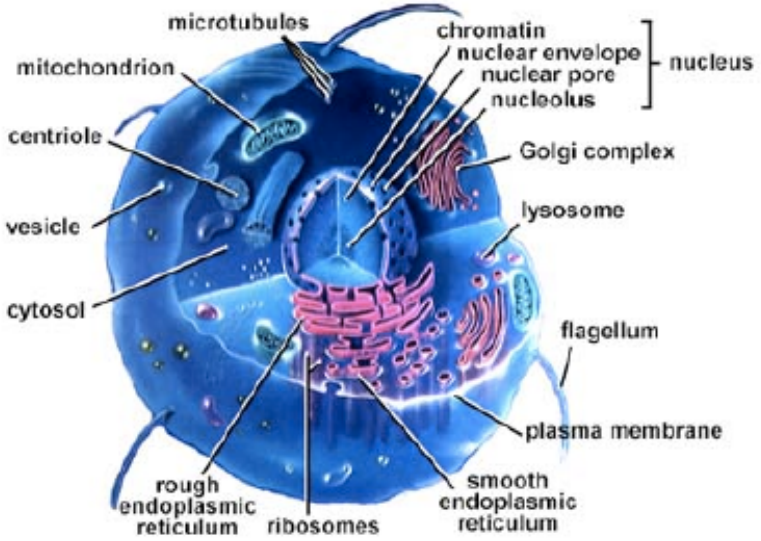
Application of Dynamic Programming

Biological Sequence Alignment

Basics of
molecular biology

Life as we know it: The Cell



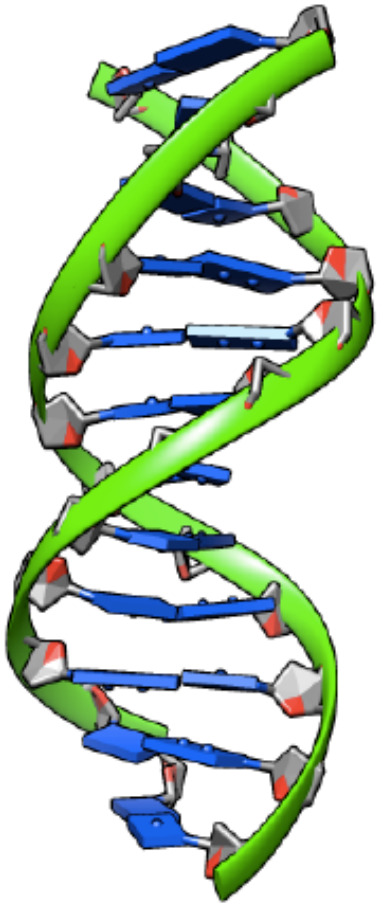


Prokaryote cell	**Eukaryote cell**
Single Cell	Single or multi cell
No nucleus	Nucleus
No organelles	Organelles
One piece of circular DNA	Chromosomes
No mRNA post transcriptional modification	Exon–intron splicing

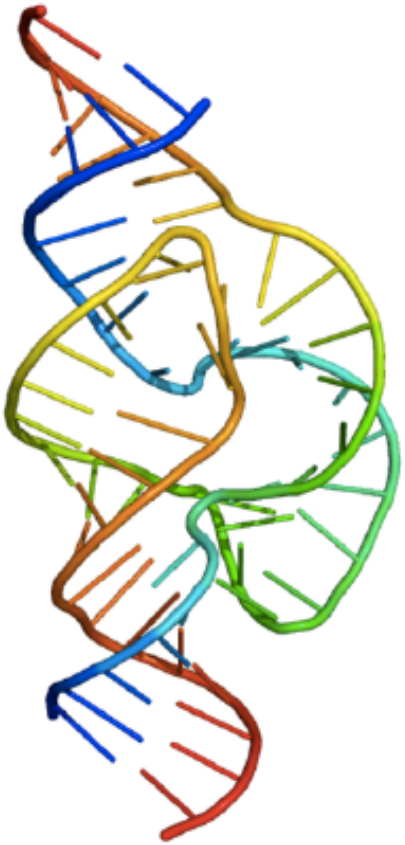
Cell Information and Machinery

- A cell stores all the information needed to replicate itself
- Almost every cell in the human body contains the same set of genes
- What differentiates cells in your body?
  - Not all genes are expressed at the same time in the same way in all cells
- A cell is a machinery
- It collects and manufactures its own components
- It carries out its own replication
- It kicks the start of its new offspring
- Life inside a cell:
  - https://www.youtube.com/watch?v=wJyUtbn0O5Y

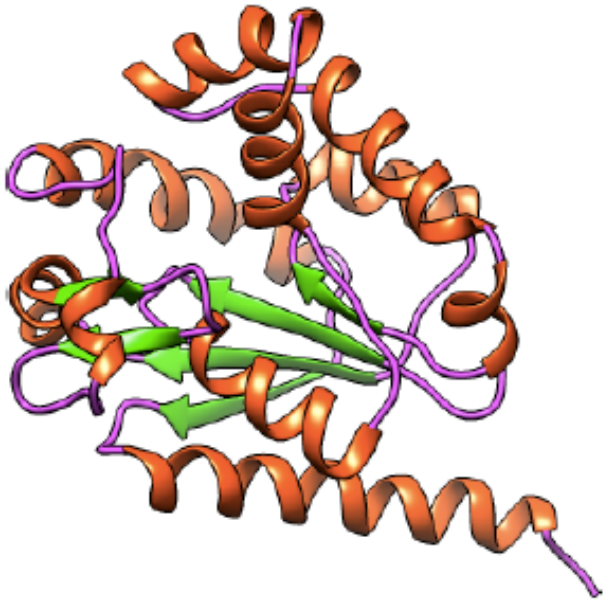
The Three Life-Critical Molecules



DNA

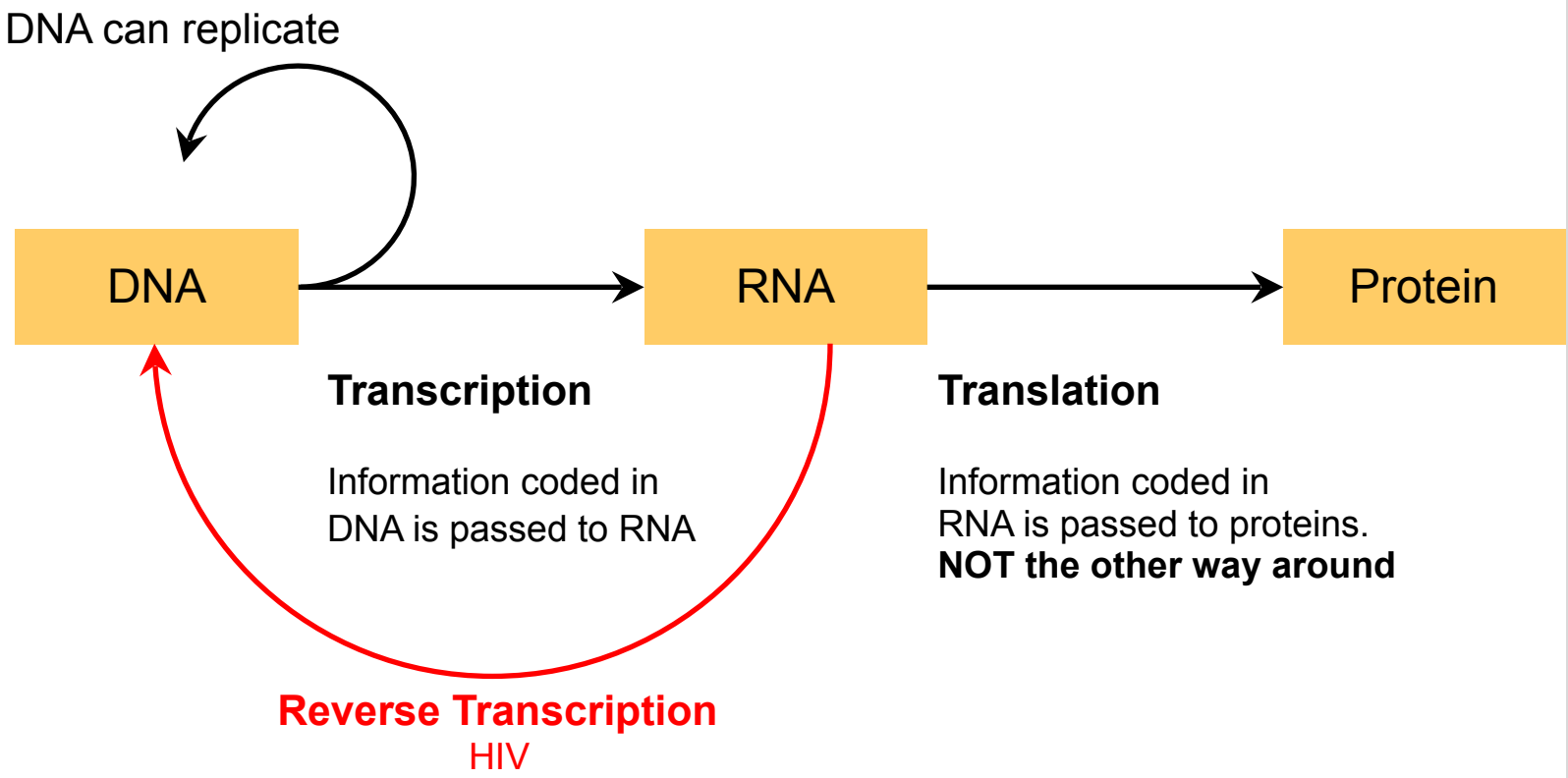


RNA



Protein

Central Dogma of Molecular Biology



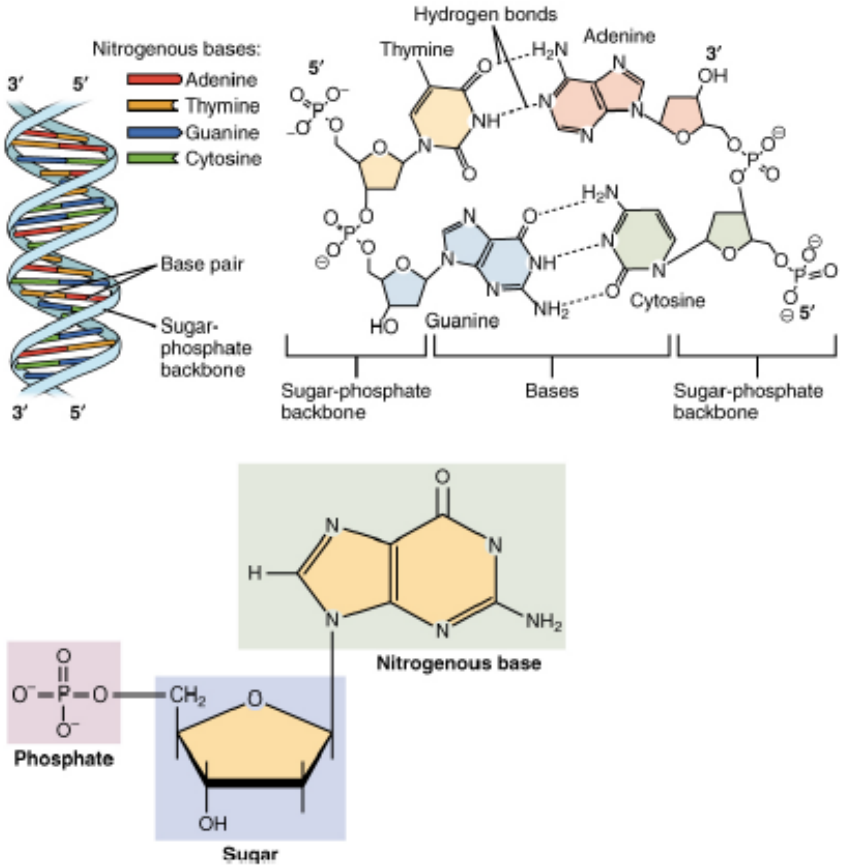
Genotype

Phenotype

Information Flow

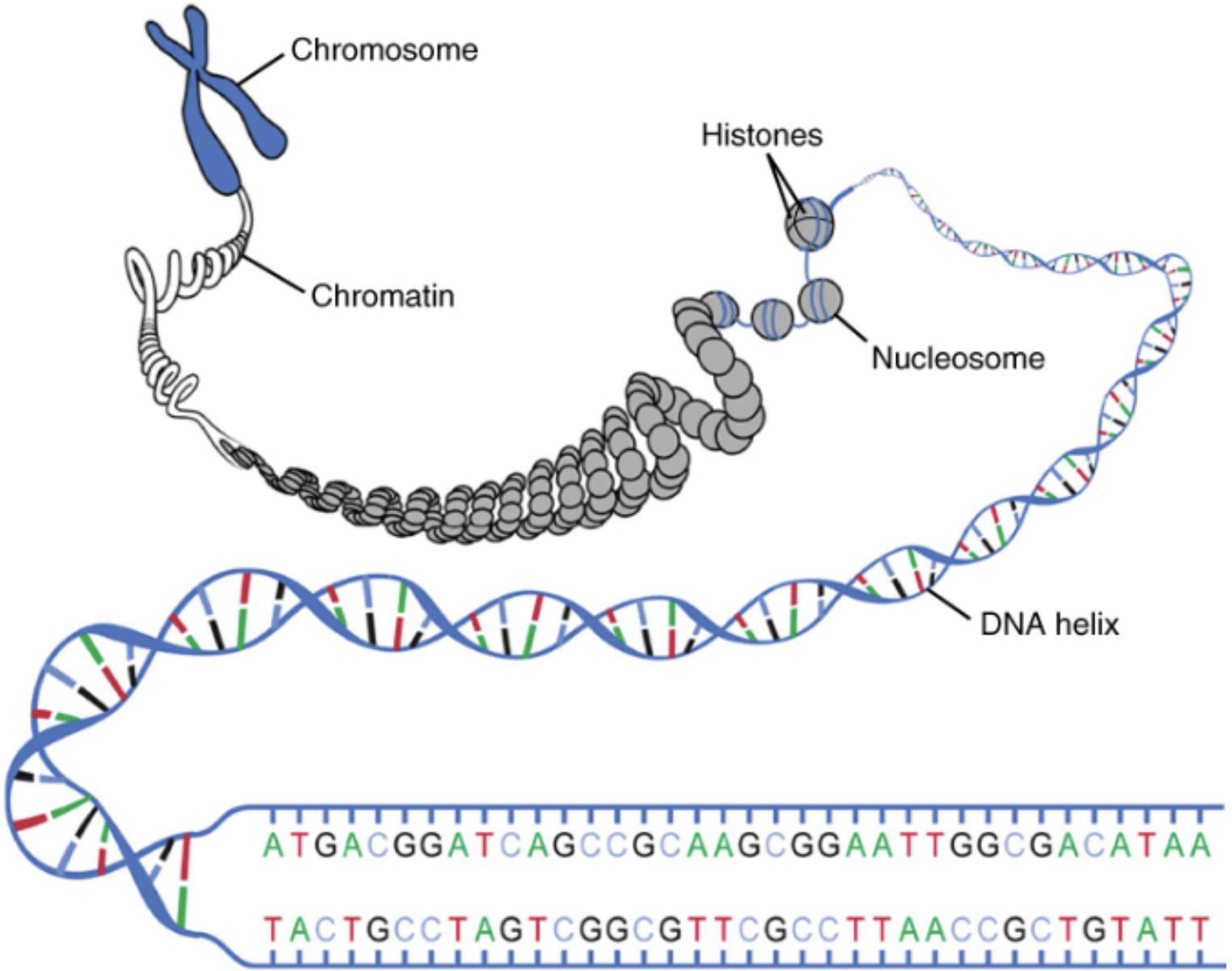
- Information flows from DNA through RNA to Synthesize Proteins in cells
- DNA
  - Holds information on how the cell works
- RNA
  - Acts to transfer short pieces of information to different parts of the cell
  - Provides templates to synthesize into proteins
- Proteins
  - Form the body's major components (hair, skin, etc.)
  - Often referred to as the workhorse of the cell

DNA



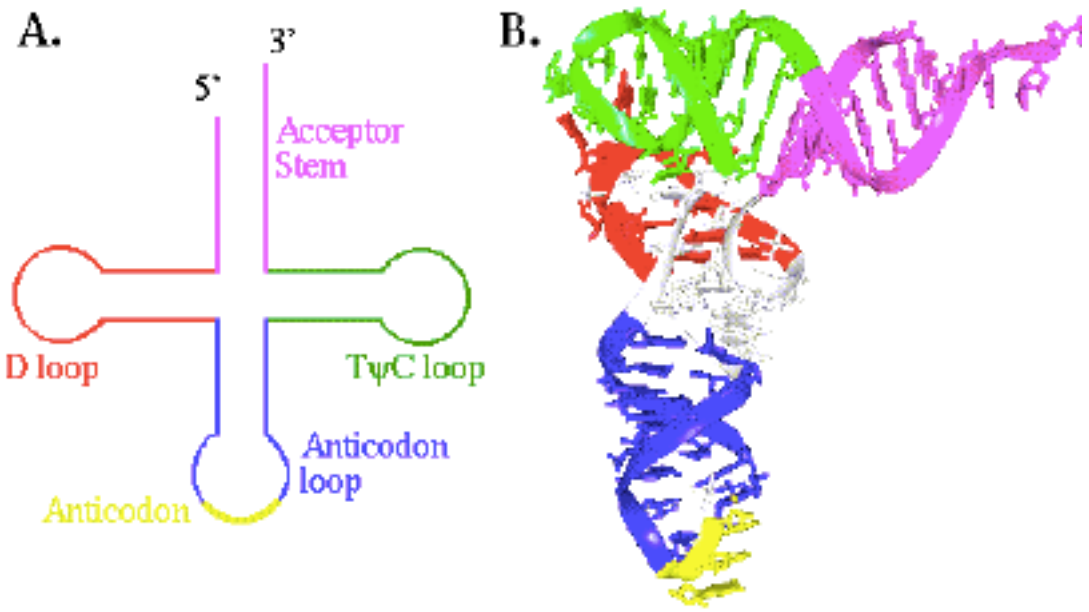
The four fundamental units of DNA are: Adenine (A), Guanine (G), Thymine (T), and Cytosine (C) They pair up on complementary strands: A-T and C-G. Like a four-letter alphabet.

DNA


Chromosome
Histones
Chromatin
Nucleosome
DNA helix
ATGACGGATCAGCCGCAAGCGGAATTGGCGACATAA
TACTGCCTAGTCGGCGTTCGCCTTAACCGCTGTATT

RNA

- RNA is chemically similar to DNA, but T(hymine) is replaced with U(racil) and the ribose instead of deoxy-ribose
- Some forms of RNA can fold to create secondary structures
  - has implication for function
- DNA and RNA can pair with each other
  - Holds information on how the cell works



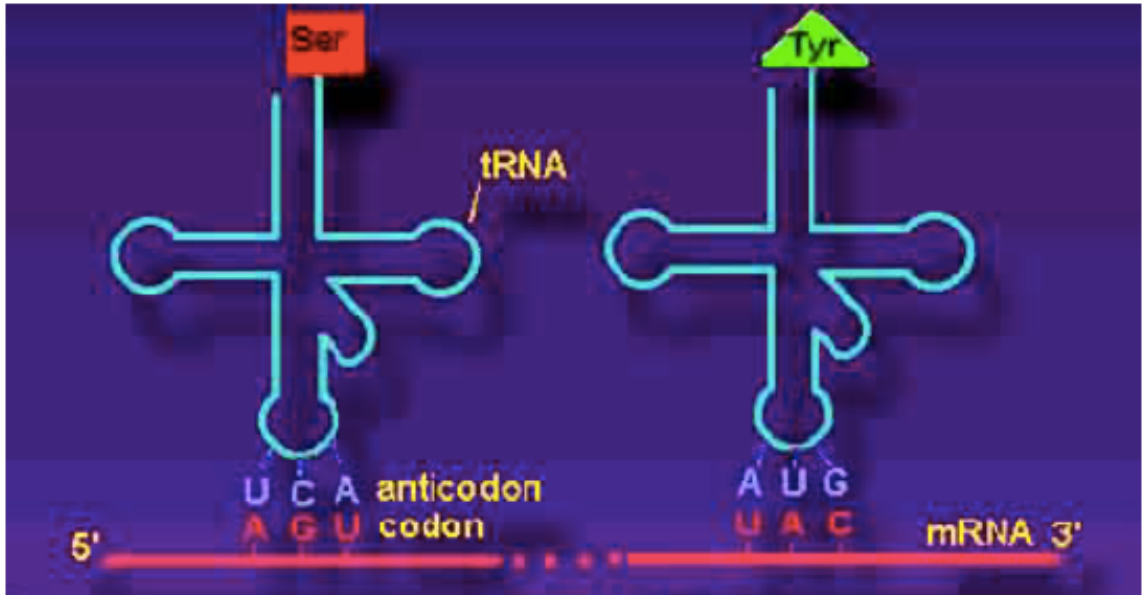
Forms of RNA

There are several forms of RNA:

- mRNA (messenger RNA)
  - carries a gene's information out of nucleus
- tRNA (transfer RNA)
  - transfer's mRNA's information onto a protein chain of amino acids
- rRNA (ribosomal RNA)
  - part of the ribosome, where proteins are synthesized

Translation

- Consecutive three Nucleotides is called a codon
- Codons map to amino acid
- Mapping often begins at start codon
- Mapping continues from 5' to 3' until stop codon



Second base

First base	U	C	A	G	Third base
U	UUU, UUC Phenylalanine F	UCU, UCC, UCA, UCG Serine S	UAU, UAC Tyrosine Y	UGU, UGC Cysteine C	U C A G
	UUA, UUG Leucine L		UAA Stop codon; UAG Stop codon	UGA Stop codon; UGG Tryptophan W	
C	CUU, CUC, CUA, CUG Leucine L	CCU, CCC, CCA, CCG Proline P	CAU, CAC Histidine H	CGU, CGC, CGA, CGG Arginine R	U C A G
			CAA, CAG Glutamine Q		
A	AUU, AUC, AUA Isoleucine I	ACU, ACC, ACA, ACG Threonine T	AAU, AAC Asparagine N	AGU, AGC Serine S	U C A G
	AUG Methionine M start codon		AAA, AAG Lysine K	AGA, AGG Arginine R	
G	GUU, GUC, GUA, GUG Valine V	GCU, GCC, GCA, GCG Alanine A	GAU, GAC Aspartic acid D	GGU, GGC, GGA, GGG Glycine G	U C A G
			GAA, GAG Glutamic acid E		

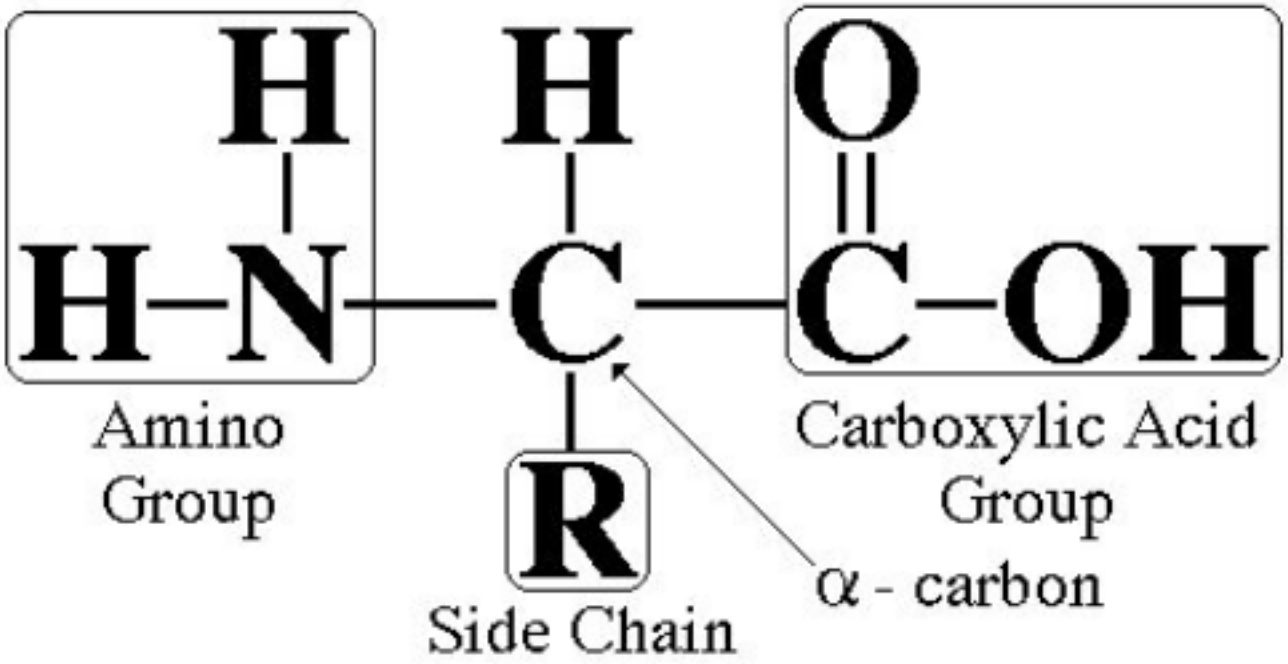
Ribosome translates RNA into protein

Venki Ramakrishnan @MRC

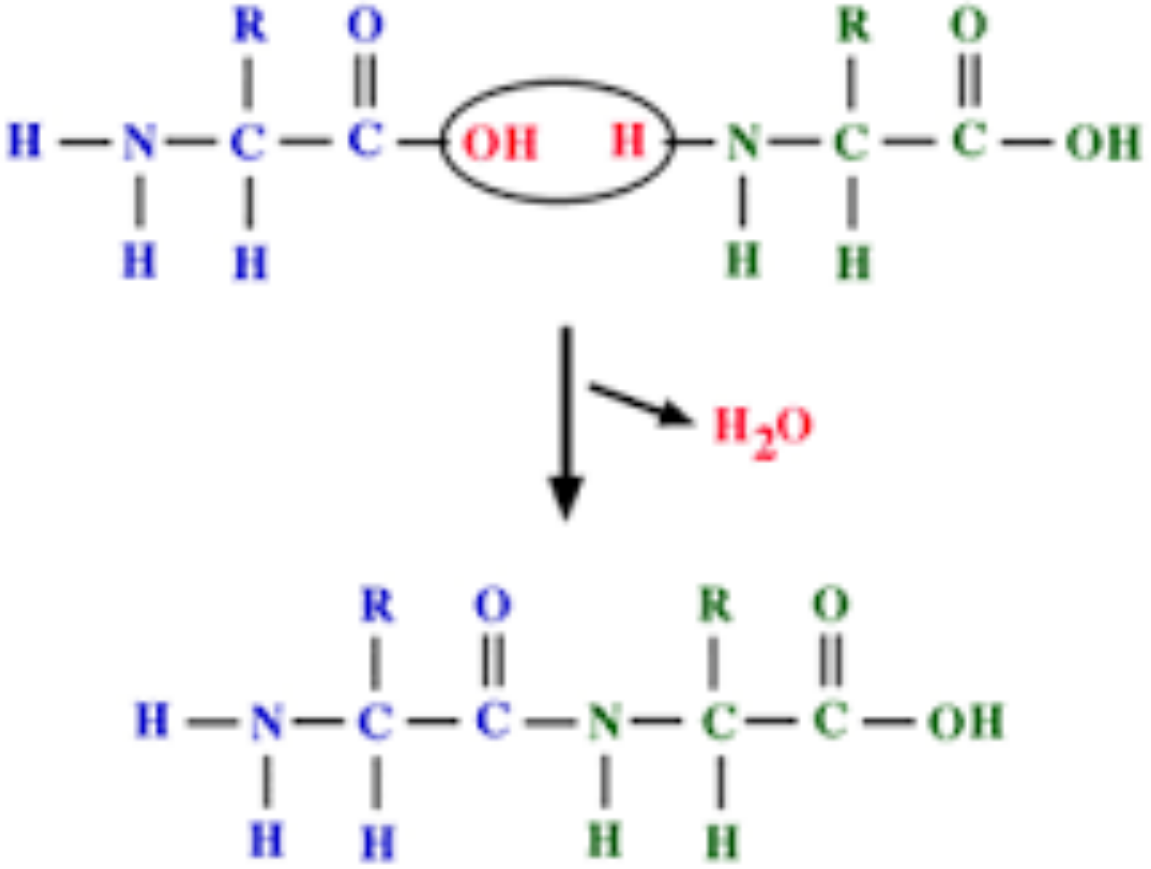
Nobel Prize in Chemistry 2009

https://www.youtube.com/watch?time_continue=123&v=q_n0Ij3K_Ho

Amino Acid


H
H
O
H–N
C
C–OH
Amino Group
Carboxylic Acid Group
R
α - carbon
Side Chain

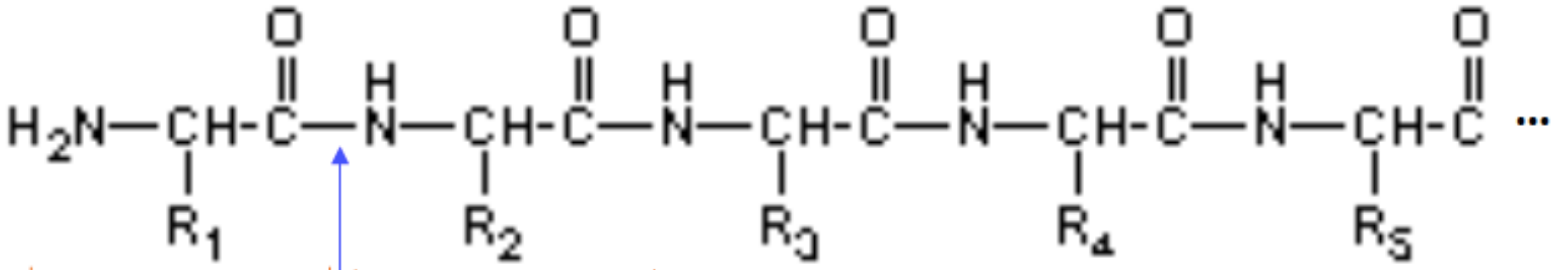
Condensation Reaction



Protein Sequence

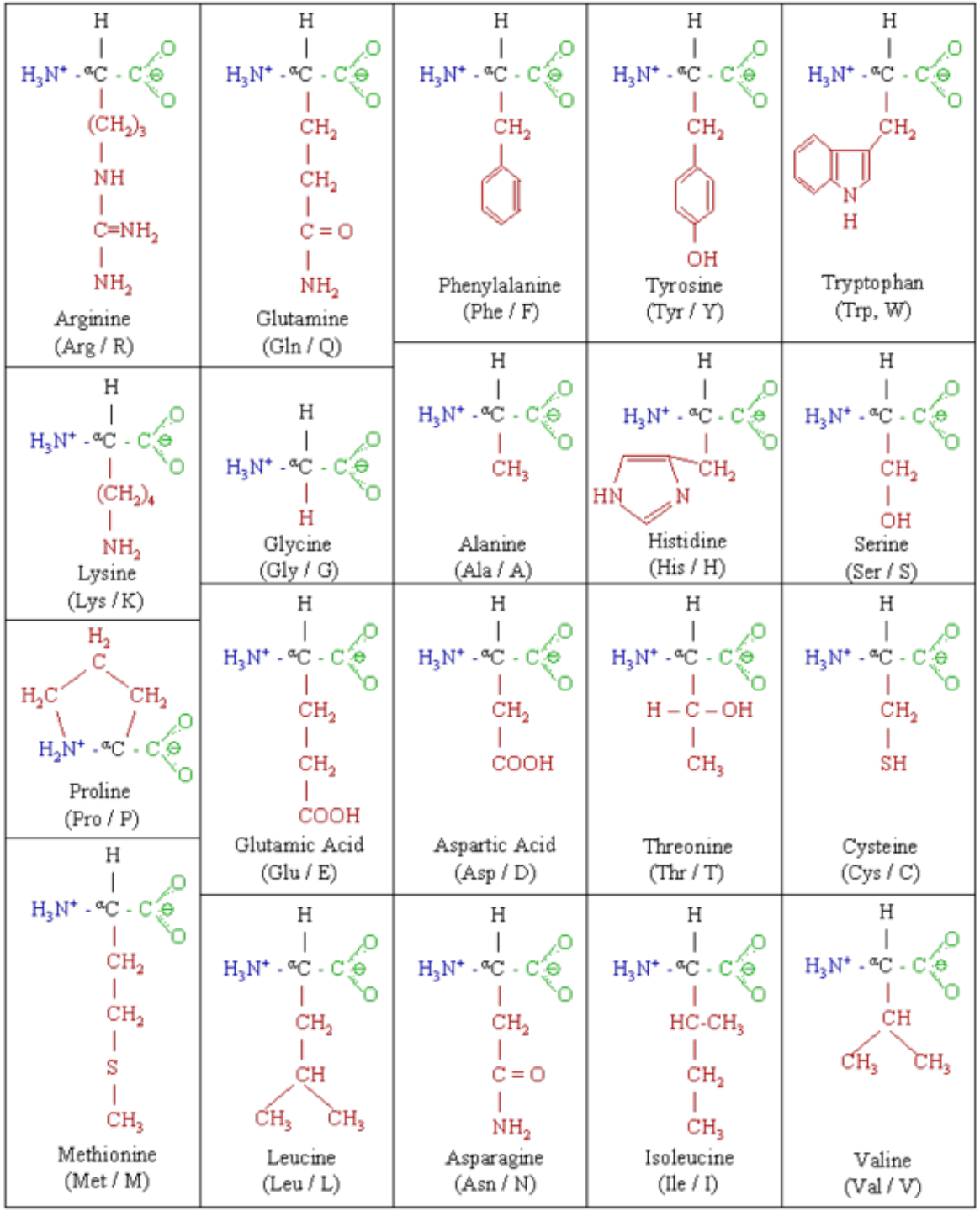
A directional sequence of amino acids/residues

N



C

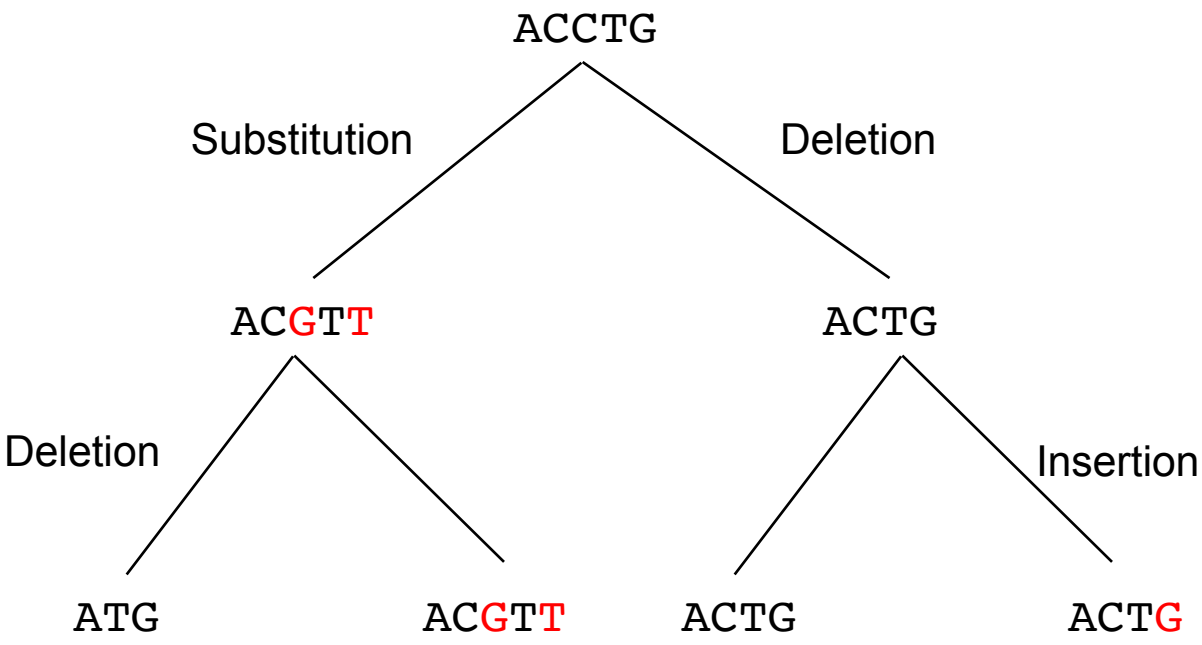
M C R I P …



20 naturally occurring amino acid residues

Evolution of
biological sequences

Sequences Evolve



Sequence Variation Through Mutation

- How sequences sequence evolve?
  - Composition changes due to point mutation (**substitutions**)
  - Sequence lengths changes due to **insertions/deletions**

- Over millions of years this result in considerable divergence between present-day sequences derived from the same ancestral sequence.

Sequence similarity

implies evolutionary relationship or **homology**

Similarity of Two Strings

S1 (16): AACTACTCCTAACACT

S2 (15): CTCCTACCTTACTTT

Inset spaces into or at the ends of to make them the same length

$S_{new}1$ (19): AACTACT-CCTAACACT--

$S_{new}2$ (19): --CTCCTACCT--TACTTT

10 matches

2 mismatches

7 spaces

10–2–7 = 1

This is called **Alignment**

After aligning S1 and S2, we compute the "**value**" of the alignment

Value of alignment = **#matches – #mismatches – #spaces**

What is the
best possible alignment?

One that maximizes
[#matches – #mismatches – #spaces]

Sequence Similarity : Optimization Problem

- Similarity of S1 and S2 is:

  arg max [#matches – #mismatches – #spaces]

  all alignments

- Can we enumerate all possible alignments?

How many alignments
are possible for two sequences of length n?

How many alignments?

- Q: How many alignments are there of two sequences of length n each?
- Let's focus on spaces, say r of them and then we sum over all possible r
- Step 1: If we insert r spaces exactly into S1, then # ways to put n chars into (n+r) placeholders

$$\binom{n+r}{n} = \frac{(n+r)!}{n!\,r!}$$

- Step 2: Put chars of S2 but avoid a resulting space opposite of space
  - We already know where the n chars of S1 are
  - We only have to consider where to put spaces from S2

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

How many alignments?

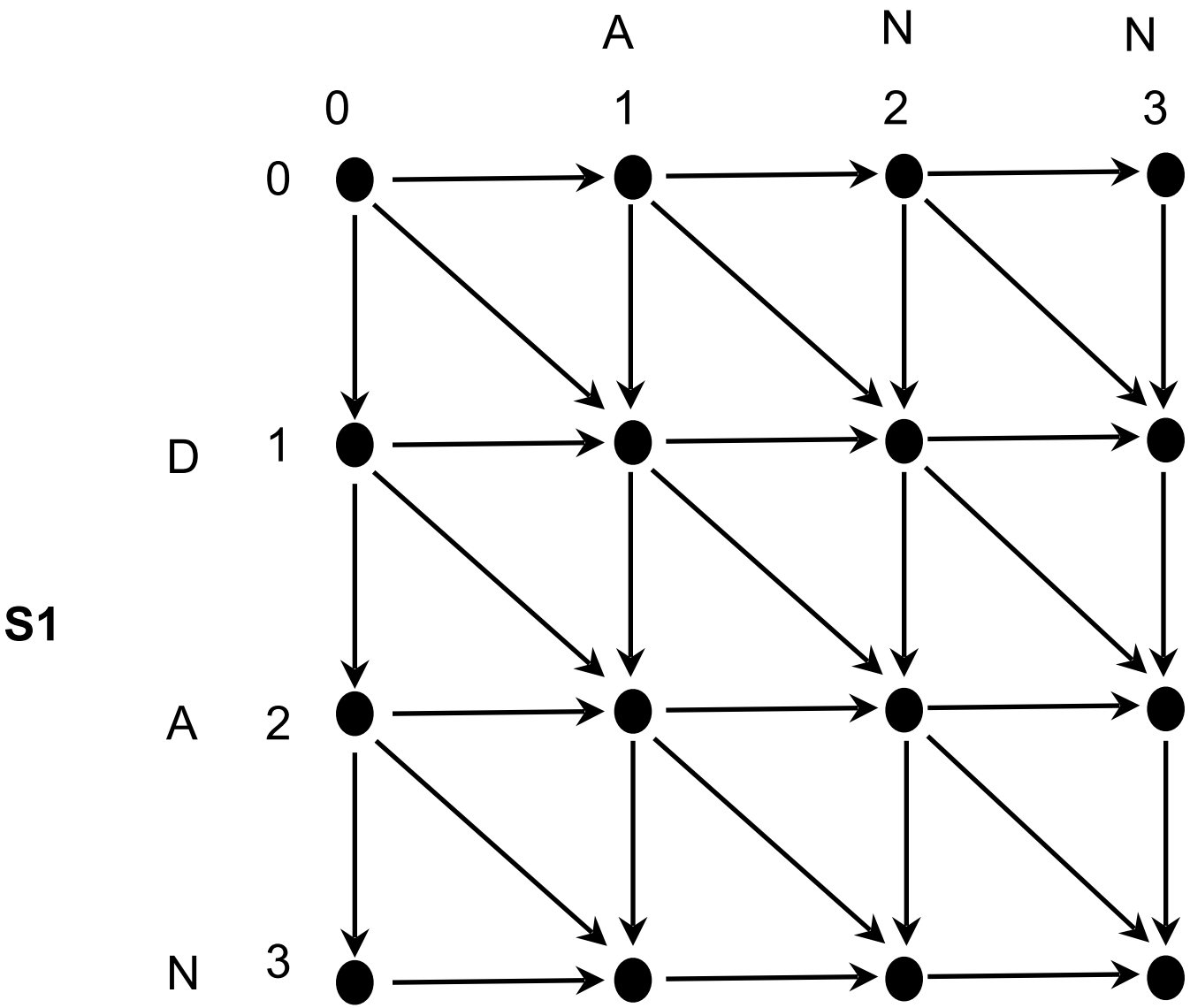
- Total number of alignments are:

$$\sum_{r=0}^{n} \binom{n+r}{n} \binom{n}{r}$$

Number of alignments are exponentially large and impossible to enumerate

Visualizing Alignment

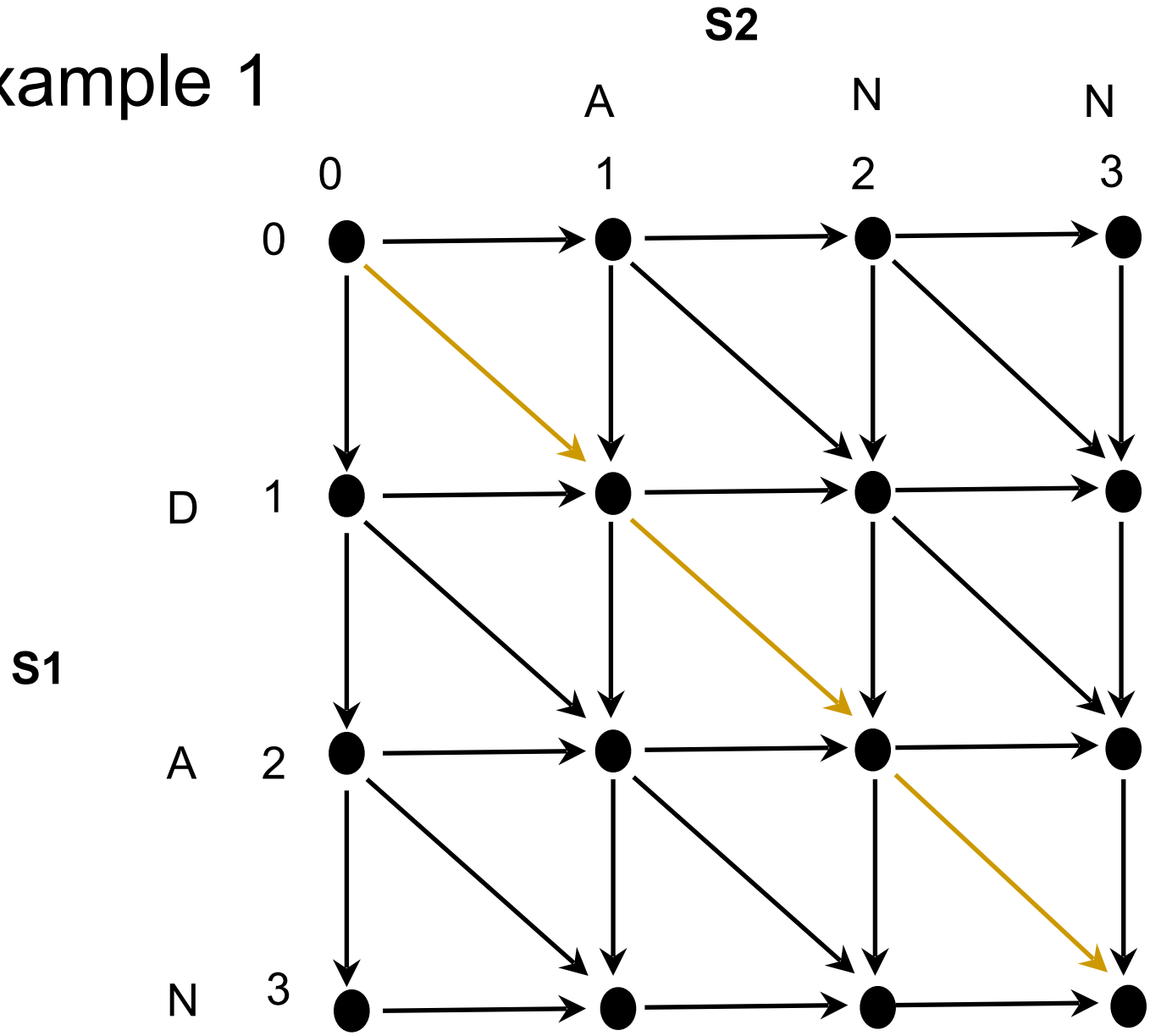
Alignment Graph



Features of Alignment Graph

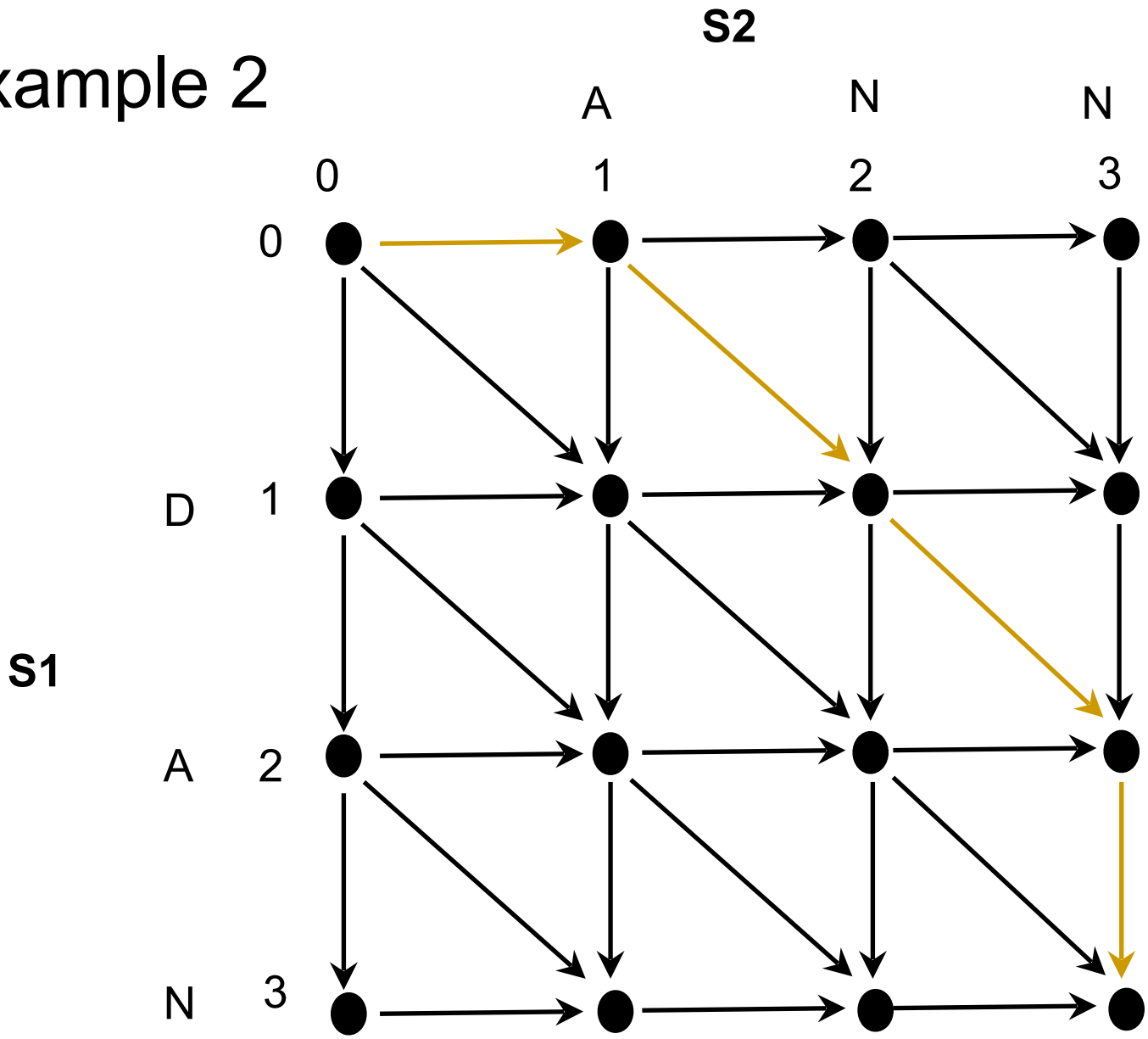
- Let's assume S1(n) is arranged in row and S2(m) is arranged in column
- Each legal path from (0, 0) to (n, m) in the alignment graph defines an alignment of S1 and S2 and each alignment defines to a distinct legal path
- Move Rules:
  - Diagonal : Put the char i of S1 opposite the char j of S2
  - Horizontal: Put the char j of S2 opposite a space of S1
  - Vertical: Put the char i of S1 opposite a space of S2
- Intuition of the Move Rule:
  - Diagonal : Using up the chars of S1 and chars of S2
  - Horizontal: Using up the chars of S2 but nothing from S1
  - Vertical: Using up the chars of S1 but nothing from S2

Example 1



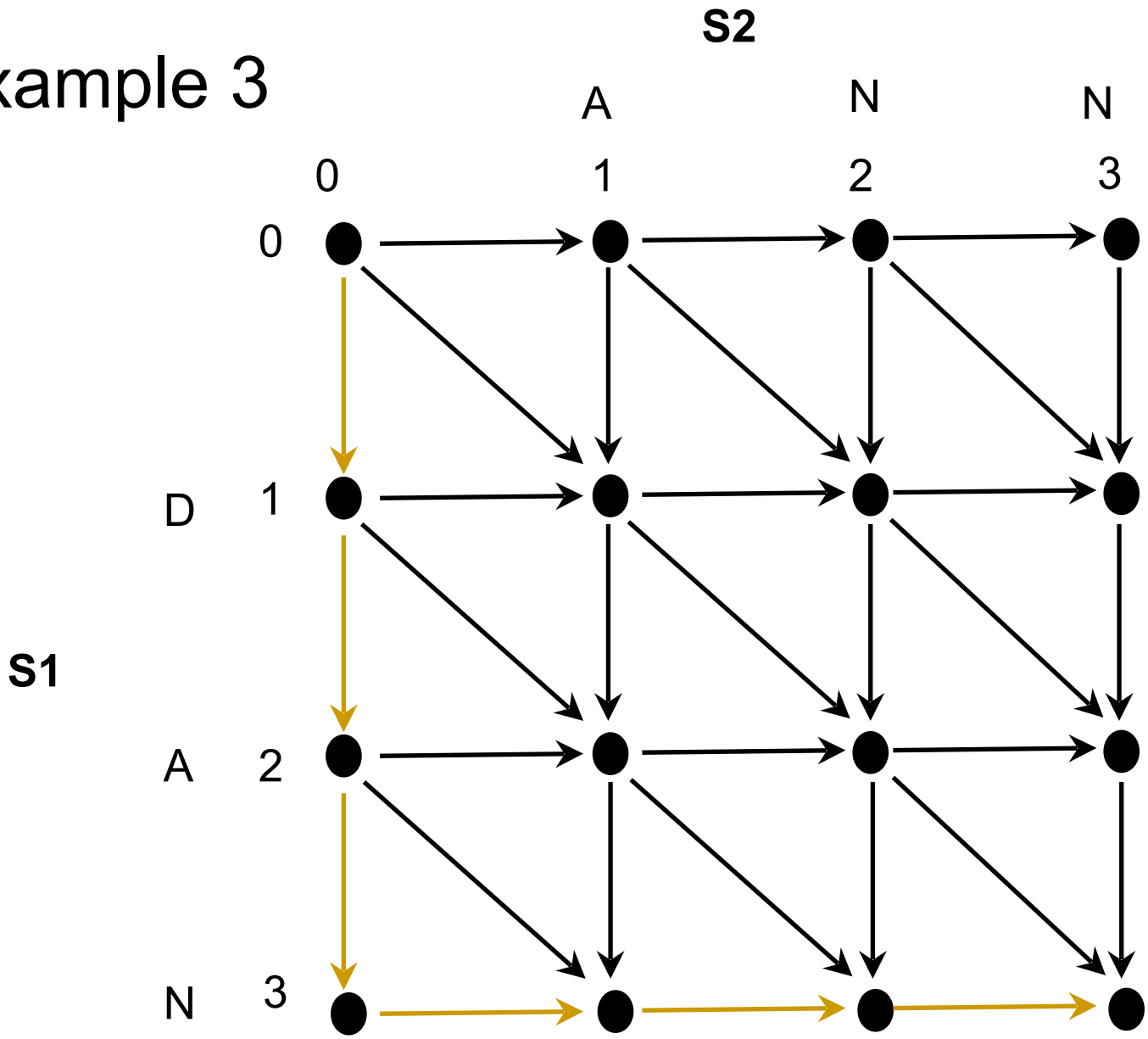
S1: DAN
S2: ANN

Example 2



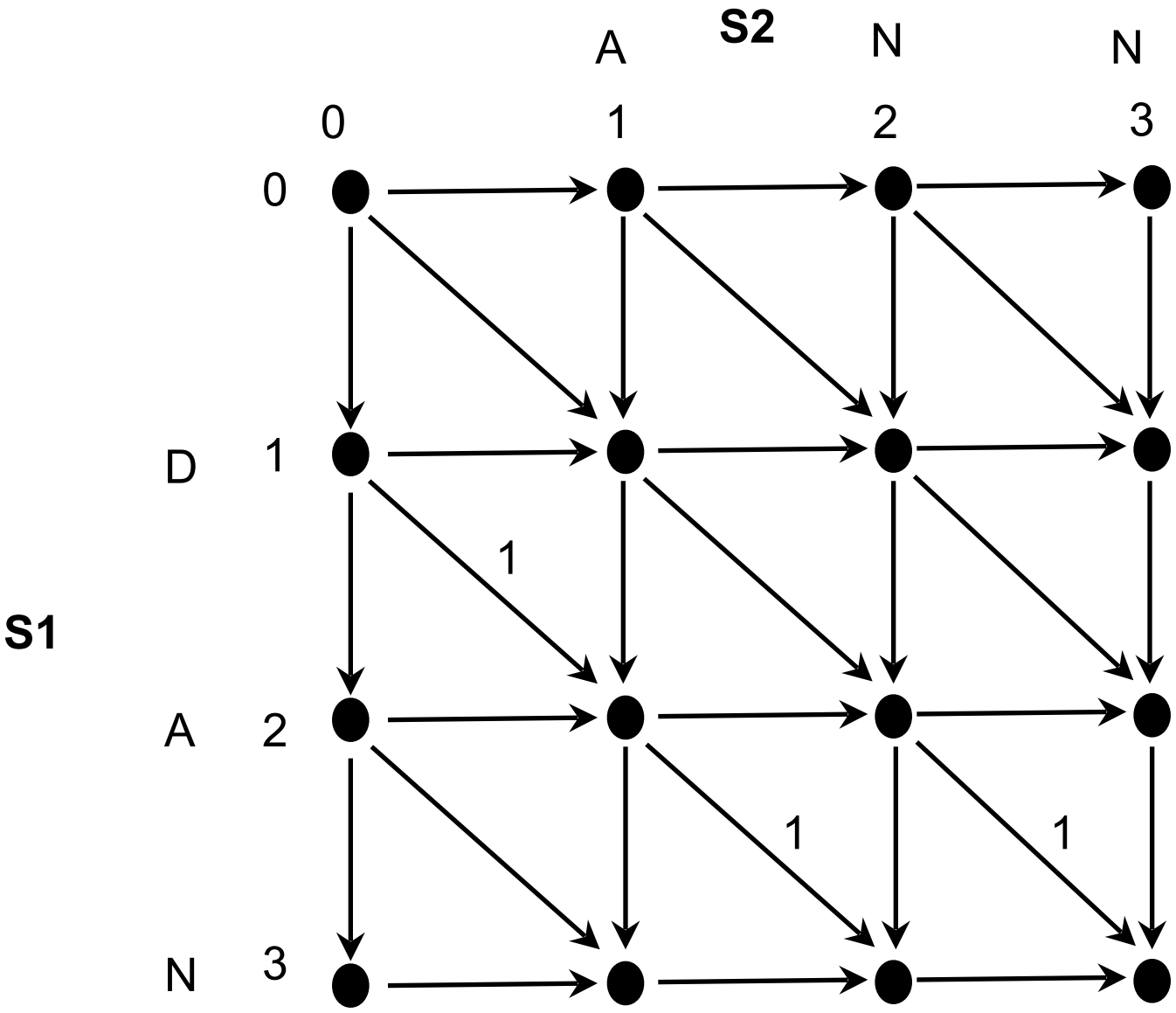
S1: -DAN
S2: ANN-

Example 3

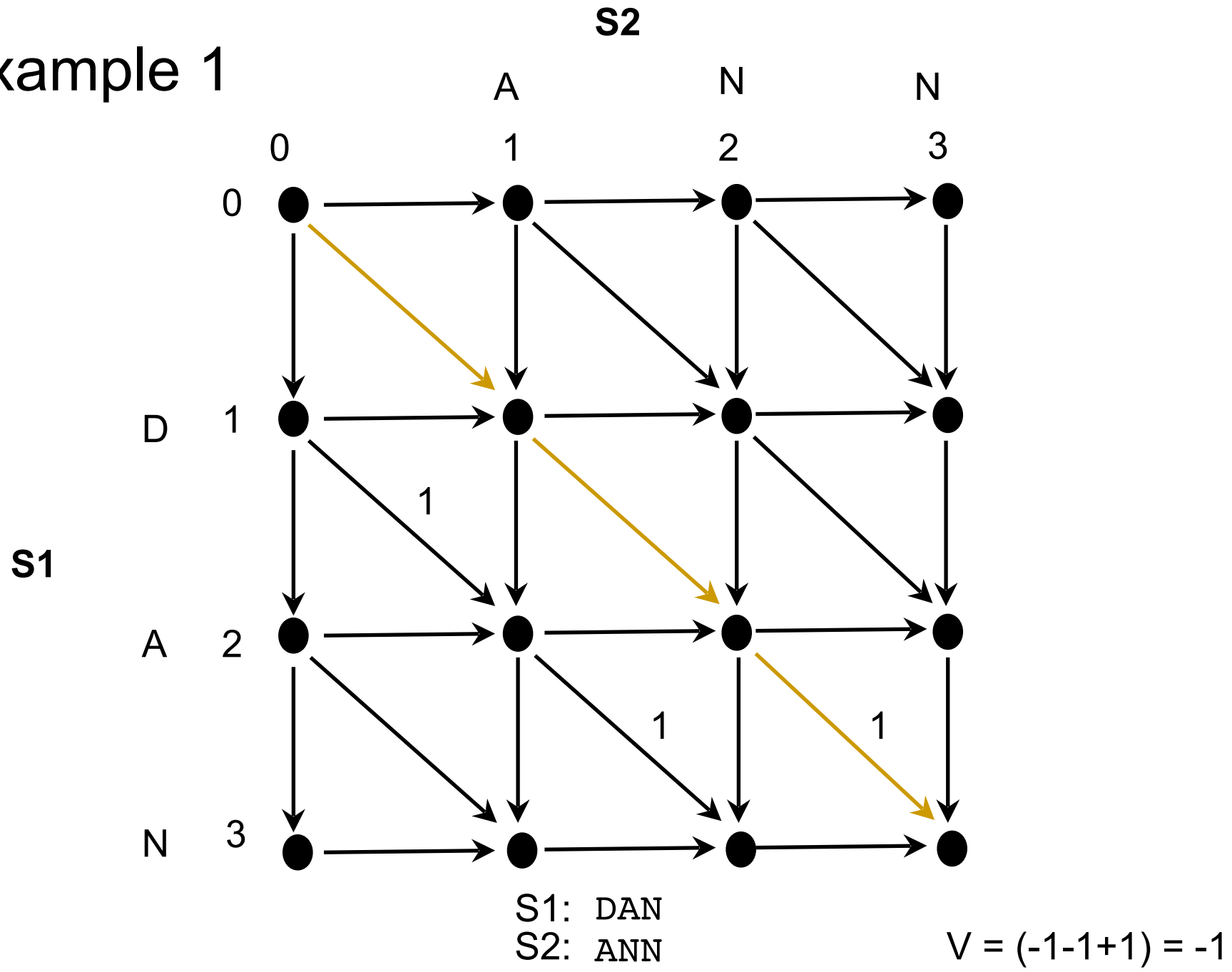


S1: DAN---
S2: ---ANN

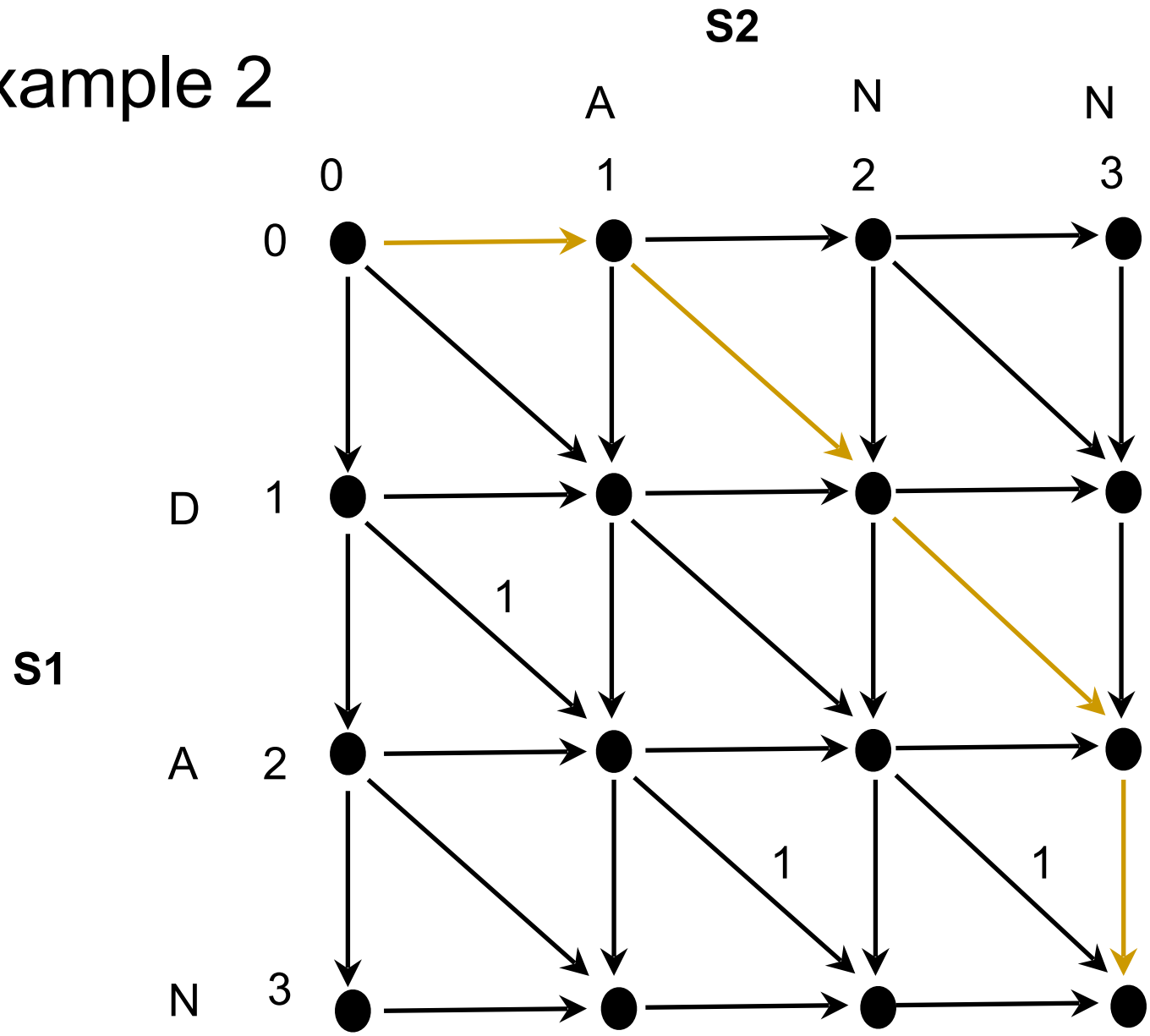
Value of Alignment



Example 1



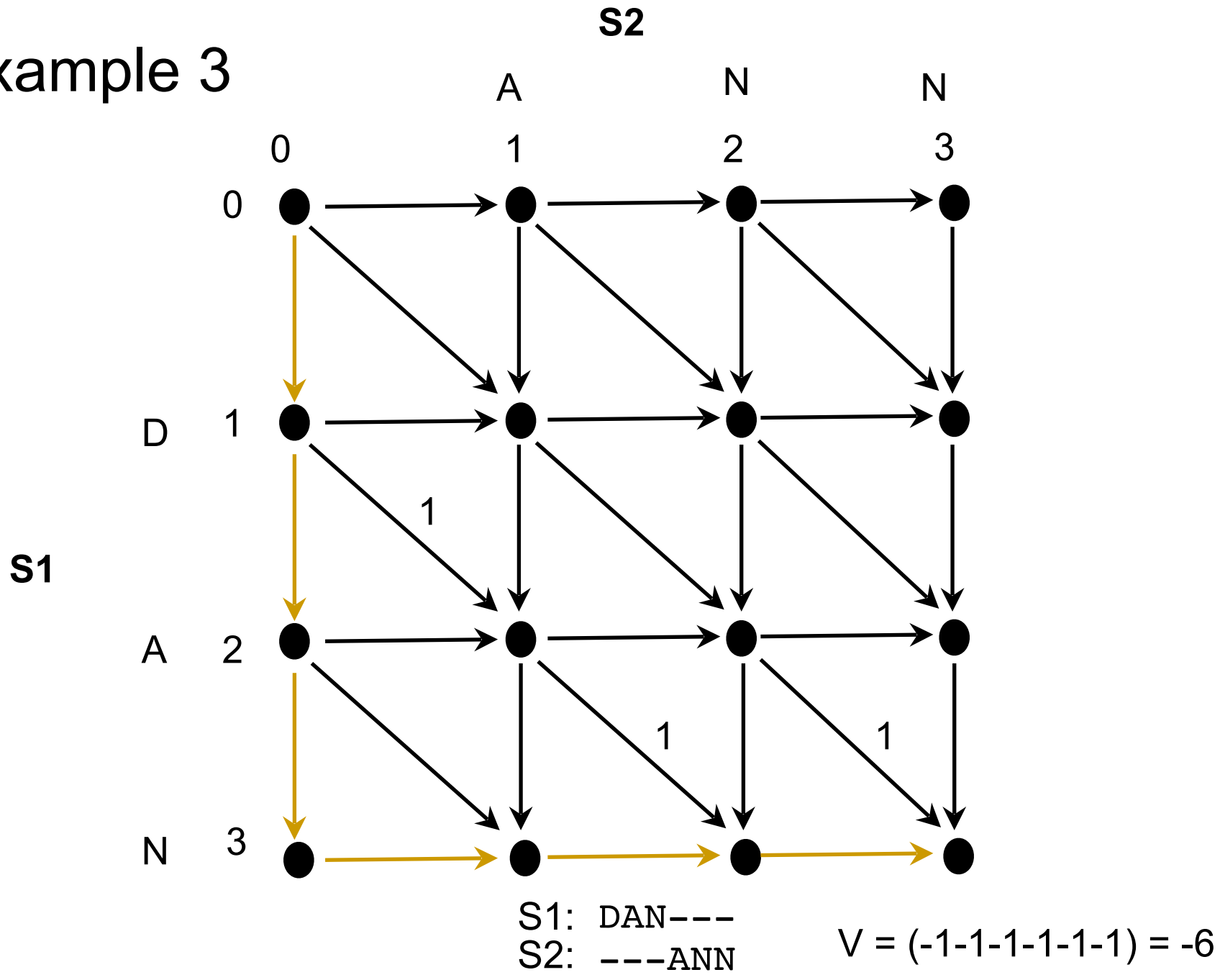
Example 2



S1: -DAN
S2: ANN-

V = (-1-1-1-1) = -4

Example 3

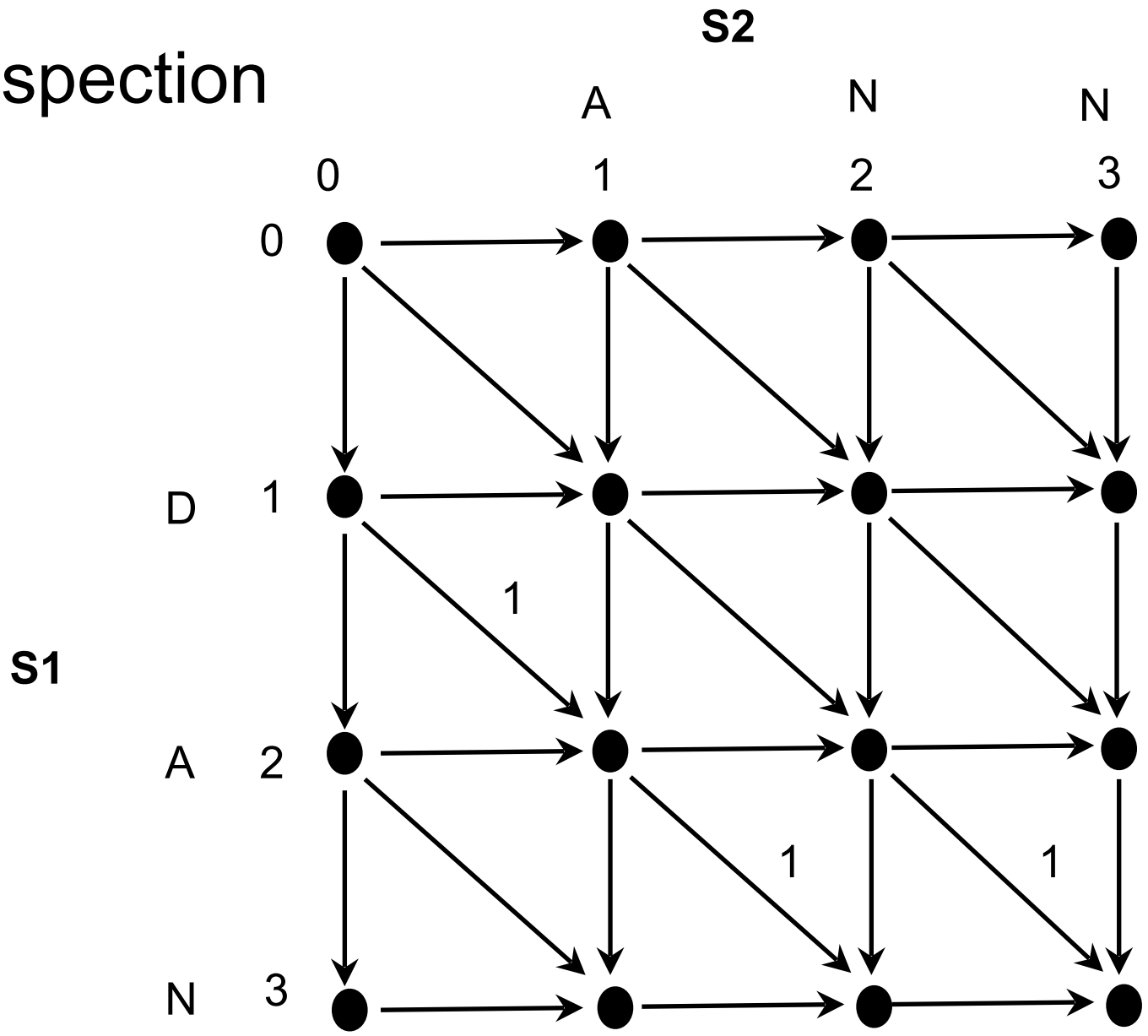


How many paths
possible from (0, 0) to (n, m)?

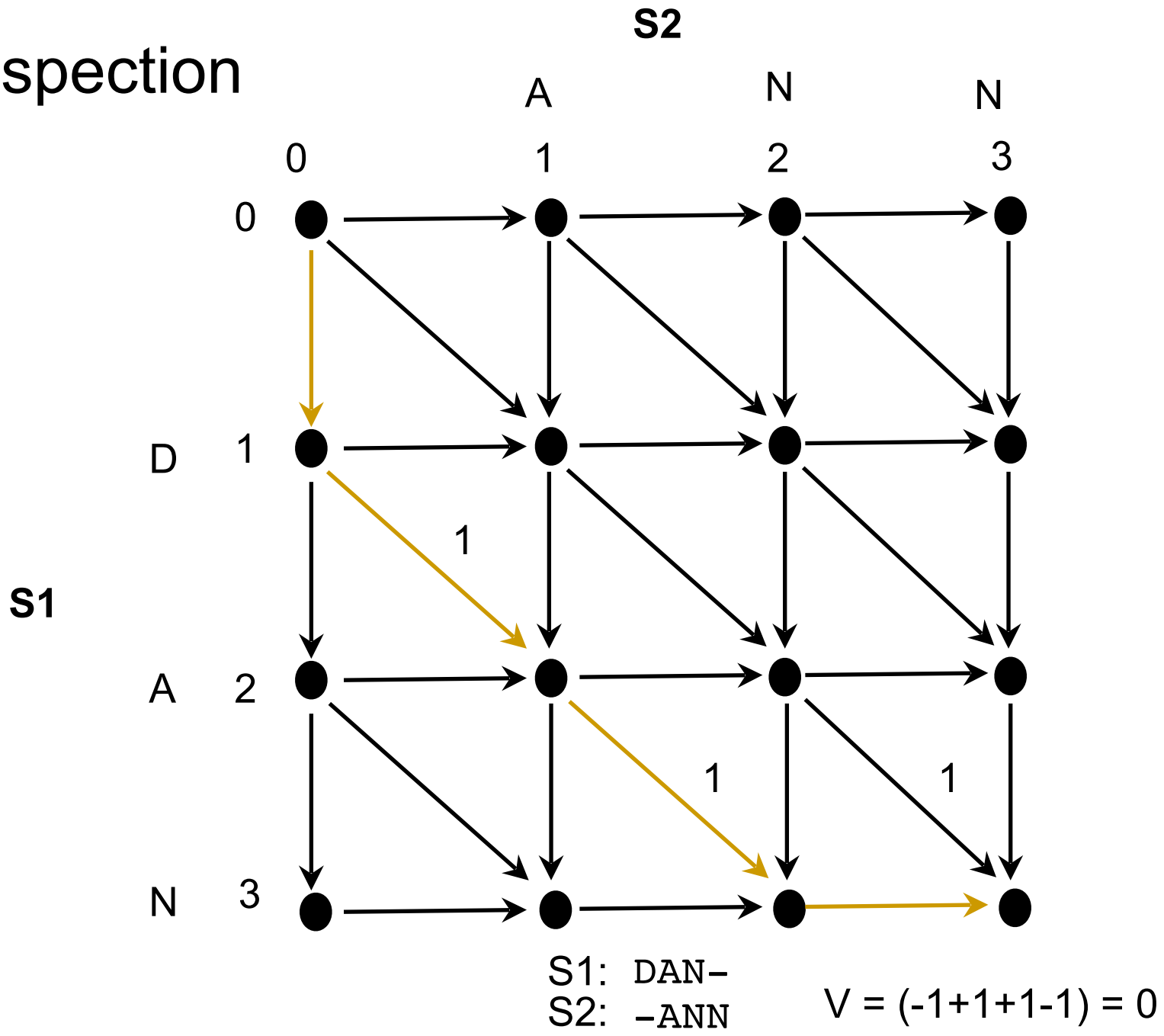
Optimal alignment

has the largest edge sum from (0, 0) to (n, m)

Visual Inspection

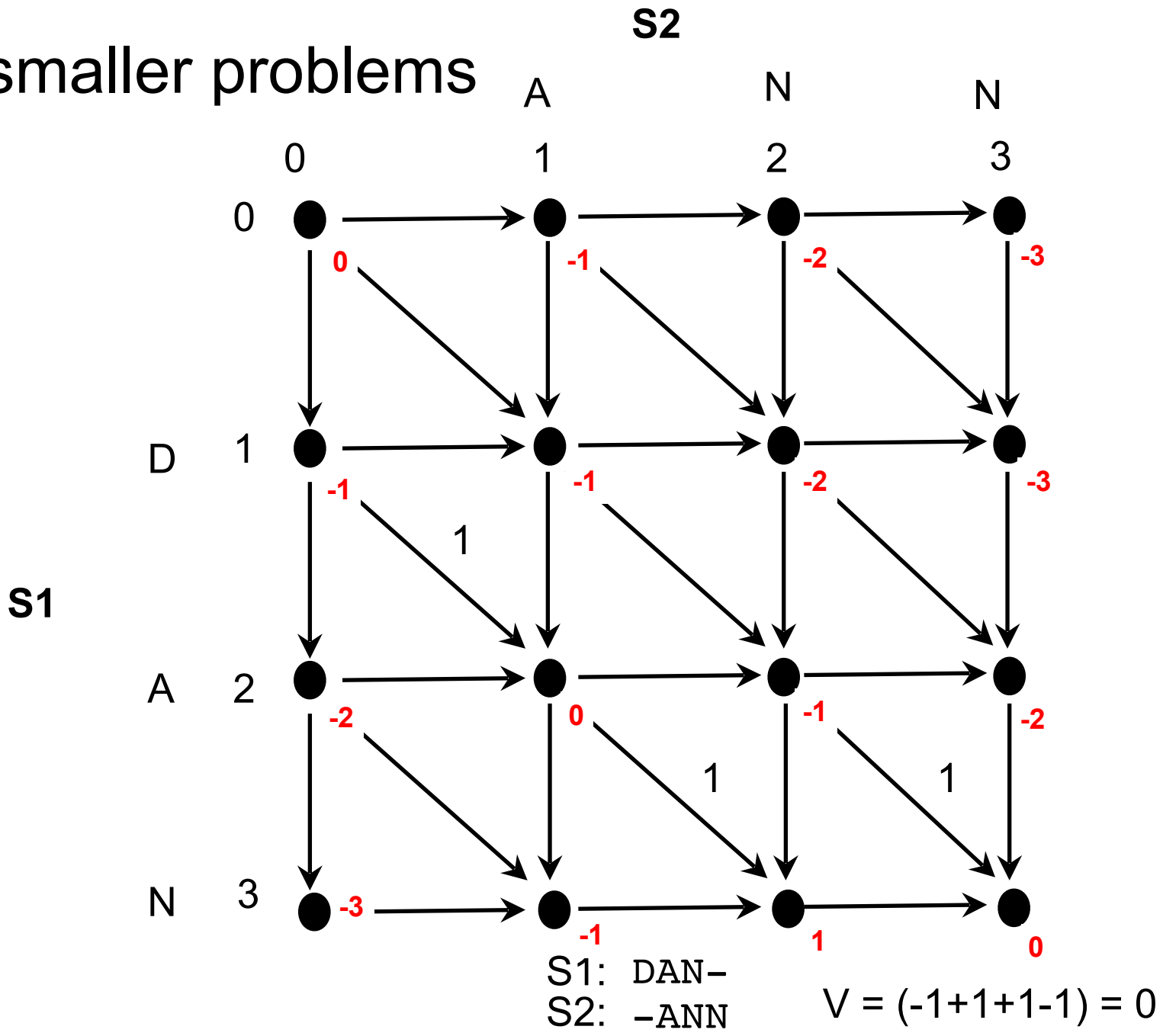


Visual Inspection



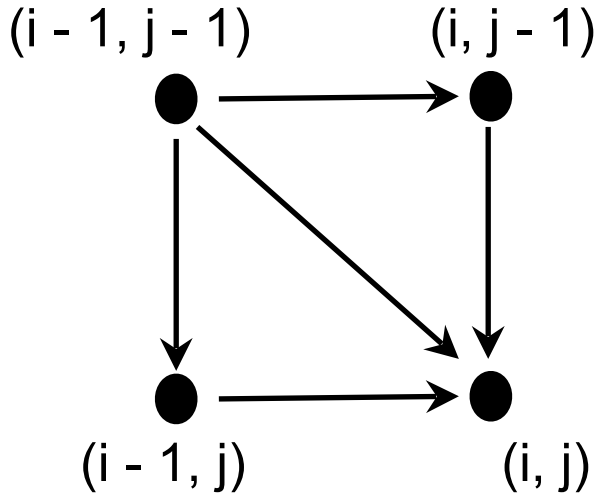
Is there a rule
for computing optimal alignment?

Solving smaller problems



Number of operations

Analysis of Number of Operations



- 3 lookups / node
- 3 additions / node
- 3 comparisons / node

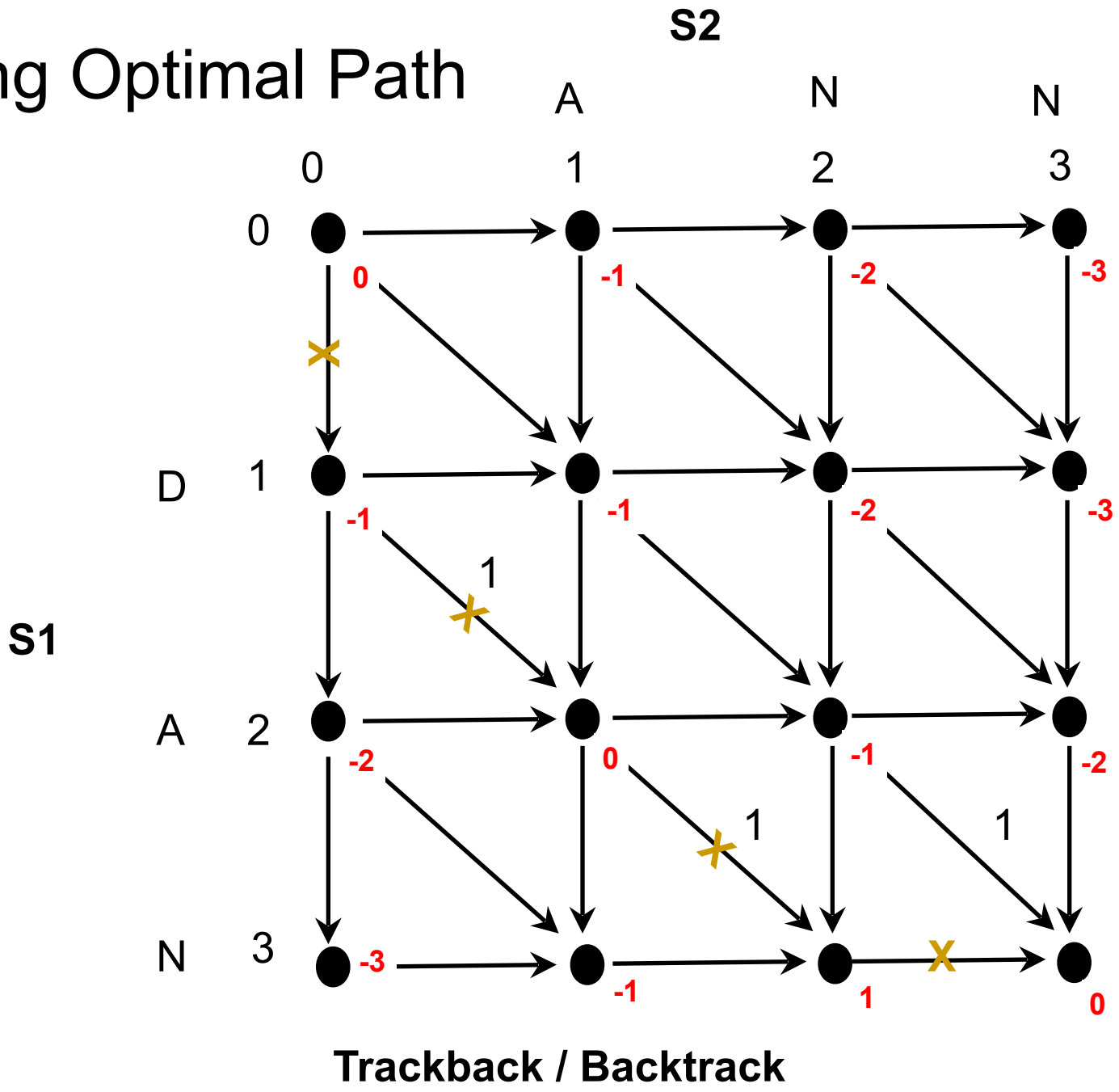
- 9 comparisons / node

- 9 operations / node no matter what the value of n and m are!

- Total operations = 9 * (n + 1) (m + 1) ~ 9**nm**

How to identify the optimal path
in the alignment graph?

Identifying Optimal Path



Dynamic Programming

Notations

- Let's say want to optimally align `S1(n)` and `S2(m)`

- For a string S, define
  - `S(k)` as the character at position `k`
  - `S[1..k]` as the prefix of S consisting of the first `k` characters of `S`

- Define
  - `V(i, j)` to be the value of optimal alignment of `S1[1..i]` and `S2[1..j]`

```
S1[1..i] .  .  .  .  .  .  .  .  .  .  .  i
S2[1..j] .  .  .  .  .  .  .  .  .  .  .  .  .  j
```

- Ultimately, we want to compute `V(n, m)`

Think Recursively!

- Let's focus on character `i` of `S1` and character `j` of `S2` and ask where these characters appear in the optimal alignment A of `S1[1..i]` and `S2[1..j]`

- There are three cases:
  - Case 1: `S1(i)` and `S2(j)` align opposite to each other in A
  - Case 2: `S1(i)` appears to the left of `S2(j)` in A
  - Case 3: `S1(i)` appears to the right of `S2(j)` in A

- These cases cover all the possibilities

Case 1

- When character `S1(i)` aligns opposite to `S2(j)` in A

```
S1[1..i] .  .  .  .  .  .  .  .  .  .  .  .  .  i
S2[1..j] .  .  .  .  .  .  .  .  .  .  .  .  .  j
```

- Either S1(i) and S2(j) are identical or not

- If S1(i) and S2(j) matches

$$V(i, j) = 1 + V(i-1, j-1)$$

- If S1(i) and S2(j) mismatches

$$V(i, j) = -1 + V(i-1, j-1)$$

Case 1

- When character `S1(i)` aligns opposite to `S2(j)` in A

```
S1[1..i] .  .  .  .  .  .  .  .  .  .  .  .  .  i
S2[1..j] .  .  .  .  .  .  .  .  .  .  .  .  .  j
```

$$V(i, j) = 1 + V(i-1, j-1) \quad \text{if } S1(i) = S2(j)$$

and

$$V(i, j) = -1 + V(i-1, j-1) \quad \text{if } S1(i) \neq S2(j)$$

Case 2 and Case 3

- When character `S1(i)` appears to the left of `S2(j)` in A, then `S2(j)` must be opposite of space

```
V(i, j) = -1 + V(i, j-1)
```

- Symmetrically, when character `S1(i)` appears to the right of `S2(j)` in A, then `S1(i)` must be opposite of space

```
V(i, j) = -1 + V(i-1, j)
```

Putting it all together

- Putting the three cases together

$$V(i, j) = \text{MAX} \begin{cases} 1 + V(i-1, j-1) \\ -1 + V(i-1, j-1) \\ -1 + V(i, j-1) \\ -1 + V(i-1, j) \end{cases}$$

- The **general recurrences** are

$$V(i, j) = \text{MAX} \begin{cases} \delta(S1(i), S2(j)) + V(i-1, j-1) \\ \delta(-, S2(i)) + V(i, j-1) \\ \delta(S1(i), -) + V(i-1, j) \end{cases}$$

Putting it all together

- The **general recurrences** are

$$V(i, j) = \text{MAX} \begin{cases} \delta(S1(i), S2(j)) + V(i-1, j-1) \\ \delta(-, S2(j)) + V(i, j-1) \\ \delta(S1(i), -) + V(i-1, j) \end{cases}$$

- The Base cases are

$$V(i, 0) = \sum_{0 \le k \le i} \delta(S1(k), -)$$

$$V(0, j) = \sum_{0 \le k \le j} \delta(-, S2(k))$$

Can we implement
this in a program?

Bottom-up Dynamic Programming

- Use **bottom-up** dynamic programming approach. We don't make explicit recursive calls, but only look up previously computed `V` values (memoize).

- First we do the base case: `V(i, 0), V(0, j)`

- Now, `V(1, 1)` can be determined

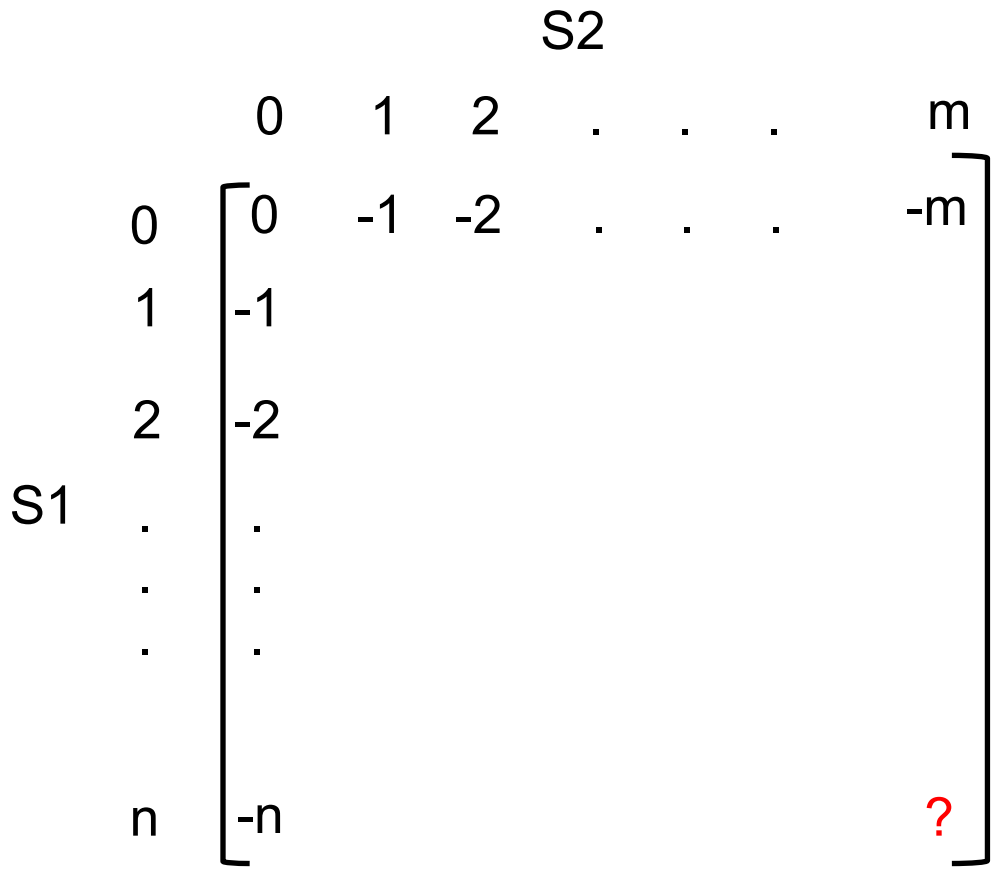
- In general, to compute `V(i, j)` we need:

  `V(i-1, j-1), V(i, j-1), V(i-1, j)`

Needleman-Wunsch
dynamic programming algorithm

DP table

- (n+1) x (m + 1) matrix of V values is a DP table



Time Complexity? ~ (nm)

BLOSUM62 substitution matrix

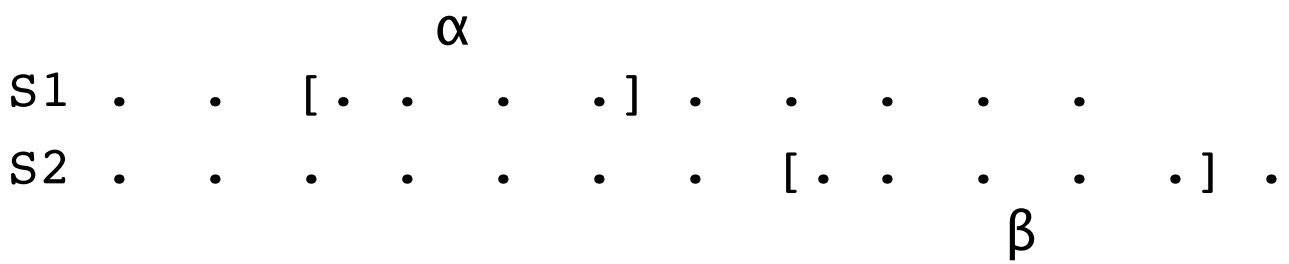
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Gap opening and gap extension

Needleman-Wunsch computes
global alignment of two sequences

Local Alignment

- Consider the following



- Given `S1(n)` &`S2(m)`

  Find a substring αof `S1` and a substring β of `S2` such that

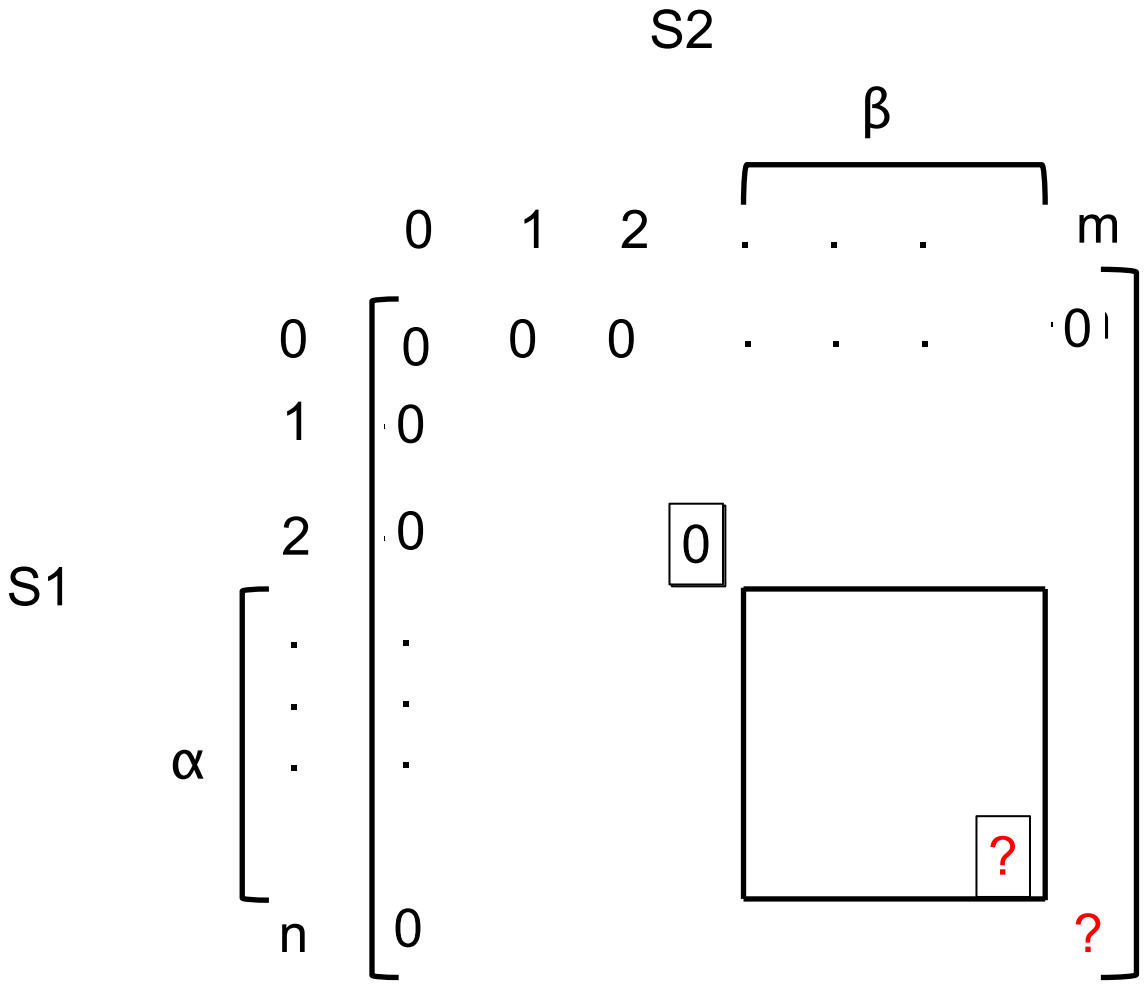
  `V(α, β)` has the largest `V` value

  over all possible pairs of substrings

  α and β that can be extracted from `S1` and `S2` respectively

Can we **efficiently**
compute local alignment?

Smith-Waterman Algorithm: Intuition



Smith-Waterman Algorithm: Notation

- The **general recurrences** are

$$V(i, j) = \text{MAX} \begin{cases} 0 \\ \delta(S1(i), S2(j)) + V(i-1, j-1) \\ \delta(-, S2(i)) + V(i, j-1) \\ \delta(S1(i), -) + V(i-1, j) \end{cases}$$

- The Base cases are $V(i, 0) = V(0, j) = 0$

- Follow **bottom-up** approach to fill DP matrix

- Trace back until we hit the first base case

Smith-Waterman Algorithm: Example

*S*1=*GTCTATCAC* *S*2=*ATCTCGTATGATG*

Match = +2
Mismatch = -1
Space = -1

		A	T	C	T	C	G	T	A	T	G	A	T	G
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	1	0	0	2	1	0	2
T	0	0	2	1	2	1	1	4	3	2	1	1	3	2
C	0	0	1	4	3	4	3	3	3	2	1	0	2	2
T	0	0	2	3	6	5	4	5	4	5	4	3	2	1
A	0	2	2	2	5	5	4	4	7	6	5	6	5	4
T	0	1	4	3	4	4	4	6	5	9	8	7	8	7
C	0	0	3	6	5	6	5	5	5	8	8	7	7	7
A	0	2	2	5	5	5	5	4	7	7	7	10	9	8
C	0	1	1	4	4	7	6	5	6	6	6	9	9	8

G T C – T A T C A C

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

Time Complexity? ~ (nm)