Database 2

Searching for sequences in databases

Cheong Xin Chan (CX)

c.chan1@uq.edu.au

Australian Centre for Ecogenomics
School of Chemistry & Molecular Biosciences
The University of Queensland

SCIE2100 | BINF6000 | Bioinformatics 1 – Introduction

Outline

- **Basic concepts of sequence searching**
  - The general approach and concept of sub-sequences (*k*-tuples or *k*-mers)
- **FastA**
  - Basic principles
  - Hashing-and-chaining algorithm
- **Basic Local Alignment Search Tool (BLAST)**
  - Differences between FastA and BLAST
  - Basic parameters of BLAST
  - BLAST algorithm using Finite State Machine

The basic concepts

- to search for **similar** sequences (in a database) to a query, **alignment** is necessary
- Needleman-Wunsch or Smith-Waterman algorithm is too slow for this purpose; faster approach was developed: **FastA** and **BLAST**
- assumption: a good local alignment should have some identical subsequences (i.e. **exact matches** of sub-sequences)
- these sub-sequences at a fixed length *k*, are referred to as ***k*-tuples** or ***k*-mers**
- usually, smaller *k* (e.g. 2–3) is used for protein sequences; larger *k* (e.g. 3–6) for DNA sequences

Example

Protein 2-tuples (*k* = 2) → **AN**, **AR**, …

DNA 4-tuples (*k* = 4) → **TAAA**, **TAAC**, …

General approach for sequence searching

1. Pre-process **query** string, e.g. generate short sub-sequences to search for
2. Quickly **align sub-sequences** with sequences in the database, keeping only high-scoring sub-alignments
3. Attempt to **join sub-alignments**, creating tentative scores
4. Perform a **thorough alignment** on high-scoring sequences

Sub-sequences at defined length ***k*** are known variously as ***k*-tuples**, ***k*-mers**, **words**, or ***n*-grams** (of length *n*)

FastA: searching similar sequences

- developed by Lipman & Pearson (1985)
- first described as FastP (P for protein); FastA (A for all) work on both nucleotide and protein sequences
- **FASTA** format – the most common text-based representation of biological sequences

David L. Lipman

William R. Pearson

FastA Server: http://fasta.bioch.virginia.edu/fasta_www2/

Key steps:

1. observe the pattern of word hits
2. identify word-to-word matches of a given length
3. mark potential matches
4. perform an optimised search using a Smith-Waterman type of algorithm

FastA: searching using hashing and chaining

Hashing

- an efficient approach for data storage and retrieval, common in computing
- in FastA, each distinct data entry (e.g. *k*-tuples) is assigned a **unique** integer (this saves storage space), i.e. the **index** (or *key*)
- relevant data (the *values*), i.e. position(s) at which the corresponding *k*-tuple is found on the query sequence, are stored in relation to each **index**
- data are structured in a *key-value* relationship

FastA: searching using hashing and chaining

Hashing: assigning indices

At k = 3 for nucleotide sequences:

- there are 4^3 = **64** possible 3-tuples
- let number $\mathbf{e(N)}$ be a distinct value for each nucleotide $\mathbf{N}$: $\mathbf{e(A) = 0}$, $\mathbf{e(C) = 1}$, $\mathbf{e(G) = 2}$ and $\mathbf{e(T) = 3}$
- a 3-tuple, represented as $x_i\, x_{i+1}\, x_{i+2}$, is assigned a value C_i:

$$C_i = e(x_i)4^2 + e(x_{i+1})4^1 + e(x_{i+2})4^0$$

$$C_i(\mathbf{AAA}) = e(\mathbf{A})4^2 + e(\mathbf{A})4^1 + e(\mathbf{A})4^0 = 0\times4^2 + 0\times4^1 + 0\times4^0 = \mathbf{0}$$

$$C_i(\mathbf{CAA}) = e(\mathbf{C})4^2 + e(\mathbf{A})4^1 + e(\mathbf{A})4^0 = 1\times4^2 + 0\times4^1 + 0\times4^0 = \mathbf{16}$$

Example

- the C_i is the **index** representation of the 3-tuple

FastA: searching using hashing and chaining

Chaining: creating a look-up table

record the position(s) on a sequence at which the *k*-tuples occurred (the *values*), and assign to the corresponding **index** (the *key*)

Example

consider **TAAAACTCTAAC** (at *k* = 3):

	Index (*key*)	**Position(s)** (*values*)
AAA →	0	2, 3
AAC →	1	4, 10
AAG →	2	–
AAT →	3	–
…	…	…
TTT →	63	–

FastA: searching using hashing and chaining

Chaining:

consider **TAAAACTCTAAC** (at $k = 3$):

Example

Position number	1	2	3	4	5	6	7	8	…
Nucleotide	T	A	A	A	A	C	T	C	…

Index of 3-tuple (C_i)	AAA	AAC	AAG	AAT	ACA	ACC	ACG	ACT	…	TAA	…
First position where the 3-tuple is found on the sequence	2	4	0	0	0	0	0	5	…	1	…

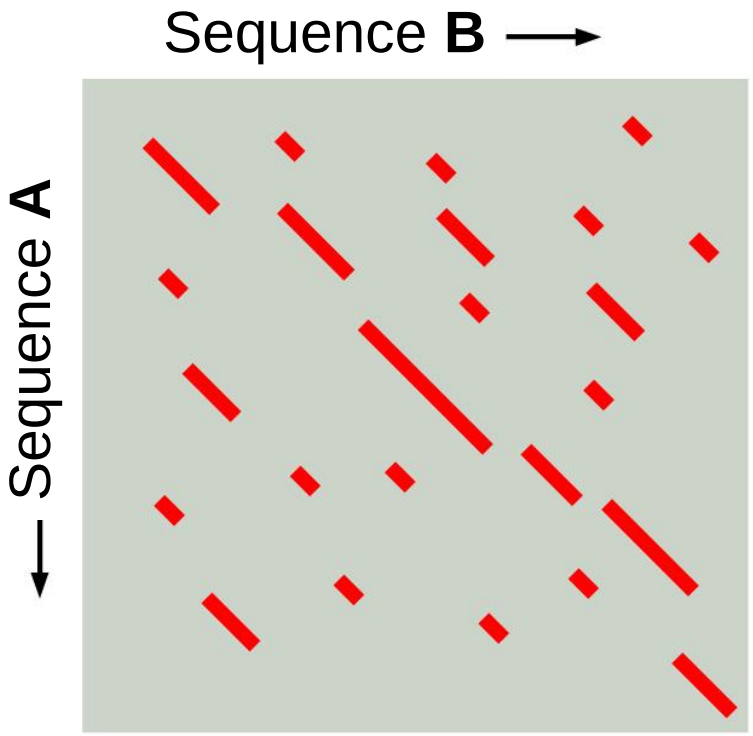
Position number	1	2	3	4	5	6	7	…
Next position the same 3-tuple is found	9	3	0	10	0	0	0	…

AAA found at position **2** then at **3** …. ; **AAC** found at position **4** then at **10** ... etc.

FastA: searching using hashing and chaining

Step 1: *observe the pattern of word hits*

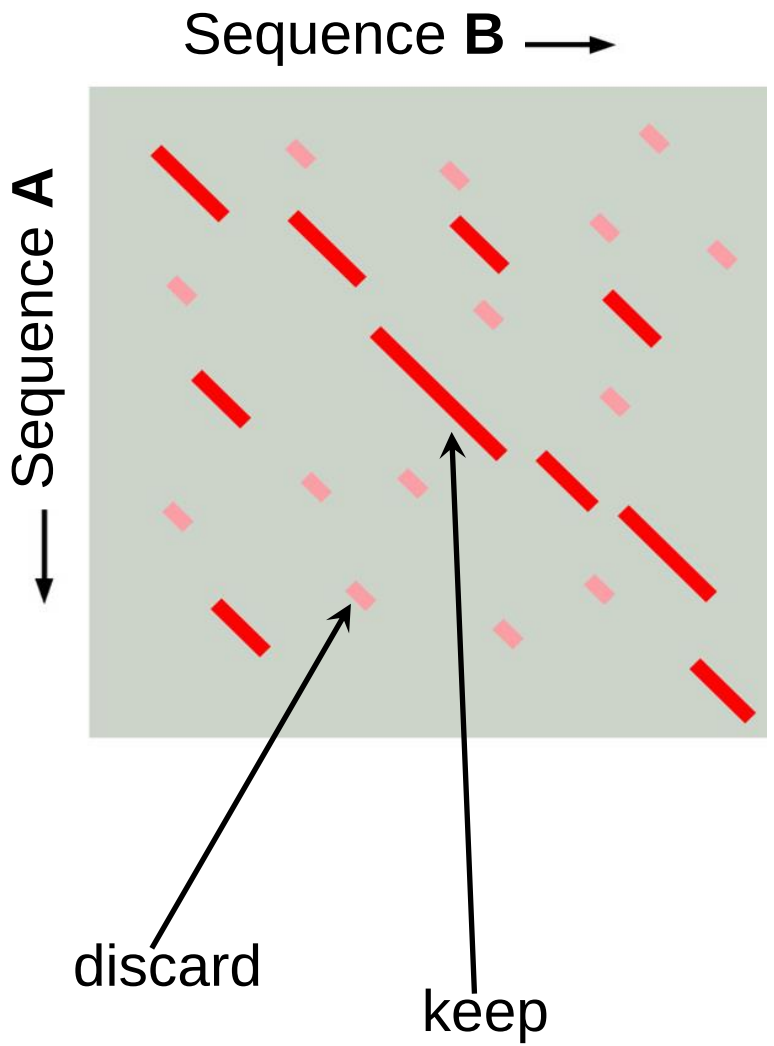
- identify perfect matches between sequence **A** and sequence **B**, using *k*-tuples *via* **hashing and chaining**
- look for high-density local regions between the two sequences; these are locally aligned regions



FastA: searching using hashing and chaining

Step 2: identify word-to-word matches of a given length

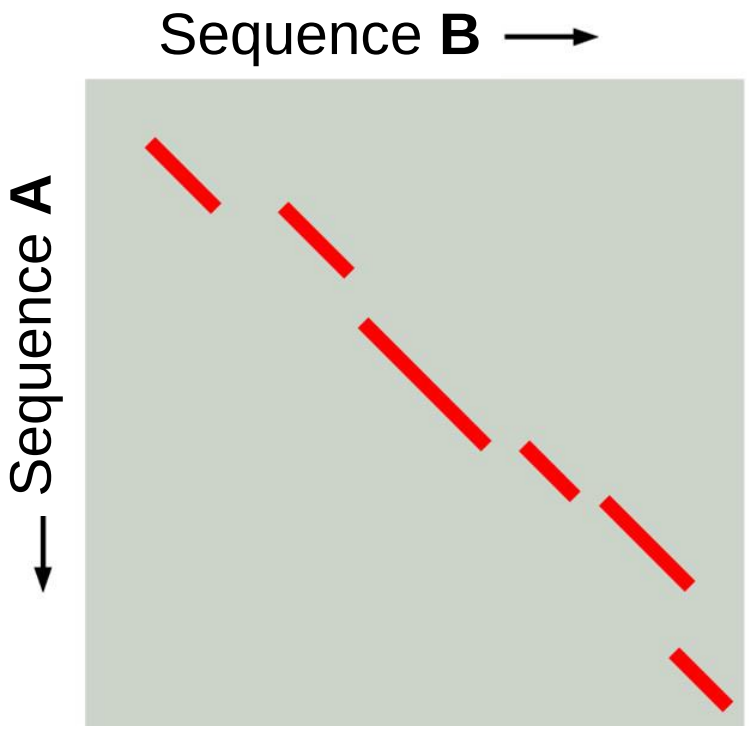
- re-score the aligned regions using a substitution scoring matrix
- keep only those contributing to the highest score, e.g. the top 10 aligned regions; discard the others



FastA: searching using hashing and chaining

Step 3: *mark potential matches*

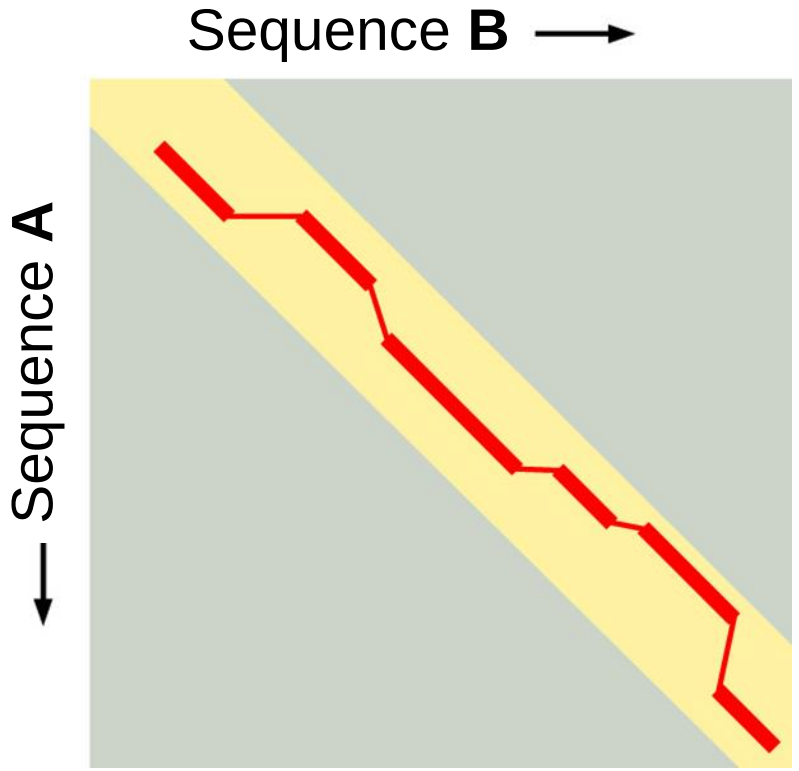
- initial regions with scores greater than a cut-off value are *joined* into an approximate alignment with gaps
- **initial similarity score** (“init***n***”) is calculated from scores of individual alignments with joining penalties
- this score is used for **preliminary ranking** of database sequences
- **init1** is the score of the single best initial region (i.e top of the rank)



FastA: searching using hashing and chaining

Step 4: *local alignment using SW algorithm*

- for suitably high-scoring database sequences, local alignment is performed between the two sequences using a Smith-Waterman algorithm
- The final score, “opt” is used for final sequence ranking; significance is calculated



BLAST: Basic Local Alignment Search Tool

- developed by Altschul et al. (1990)
- search for **High-scoring Segment Pairs** (HSPs) contained in a **statistically significant** alignment
- most commonly used tool for sequence searching in major databases: http://blast.ncbi.nlm.nih.gov/Blast.cgi
- similar approach to FastA but with **two** key differences:

FastA	**BLAST**
uses identical *k*-tuples (**exact matches**)	uses *k*-mers in the target that score above a **threshold *T*** (allowing for **non-exact matches** for protein sequences)
algorithm based on **hashing and chaining**	algorithm based on **finite state machine** (FSM), also known as **finite state automata** (FSA)

BLAST: the key parameters

Word size, i.e. *k* in *k*-mer

- typically **3** (2-4) for protein sequences, **11** for nucleotide sequences

Threshold *T* (typically an integer between 11 and 19):

- only *k*-mer (exact or inexact) matches that score **equal to or greater than *T*** (a neighbourhood score threshold) are used for seeding

Scoring (substitution) matrix

- typically BLOSUM62; the optimal scoring matrix depends on expected sequence similarity

Drop-off *X*

- the amount of drop in score that will stop the sequence extension, typically 20

BLAST algorithm

Phase 1: *remove low-complexity region or repeats in the query sequence*

```
LLQQQQAVMLQQQQLQEFYKKQQEQLHLQLLQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQHPGKQAKEQQQQQQLAAQQLVFQQ
```

Example

Low-complexity region in a sequence composes of very little variation (i.e. fewer distinct residues)

- they might yield high alignment scores but not due to biologically significance, e.g. between two non-homologous sequences (simply by chance)
- these regions in the sequence database and in the query are commonly masked before a search

BLAST algorithm

Phase 2: *identify k-mers in query with scores above threshold T for seeding*

Finite state machine (FSM)
a computational algorithm used to model a variety of phenomena that are described by a succession of states that control the behavior of a system

- given the current state and an input, the next state of the FSM can be **deterministically determined** (not probabilistic)
- given a list of *k*-mers in the query (input), the FSM can quickly produce all possible matches with scores exceeding a **threshold *T***; these *k*-mers are used for seeding (to start the alignment process)

BLAST algorithm

Phase 2: *identify k-mers in query with scores above threshold T for seeding*

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

query

Consider a 3-mer **CHH**, and ***T* = 19**

Based on BLOSUM62 matrix, each of the following transitions has a score ≥ 19:

- **CHH→CHH**: 9 + 8 + 8 = **25**
- **CHH→CHY**: 9 + 8 + 2 = **19**
- **CHH→CYH**: 9 + 2 + 8 = **19**

Example

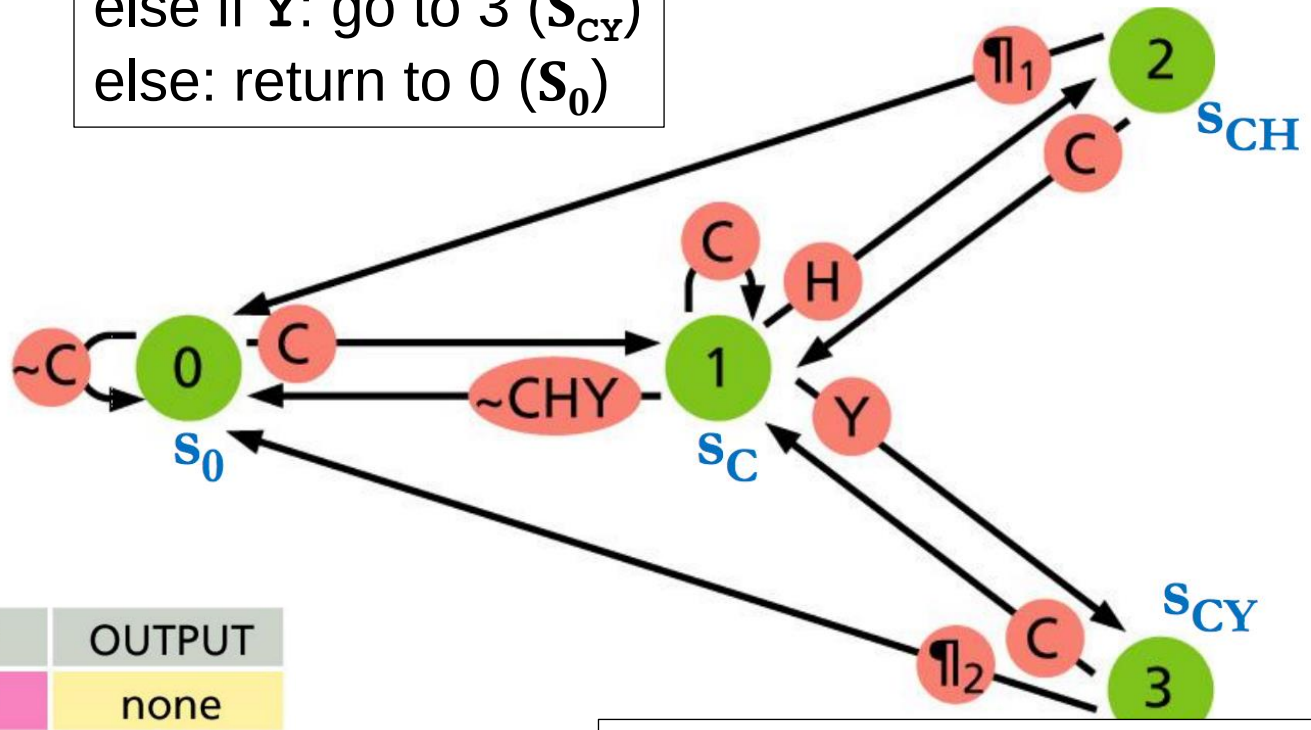
FSM

Example

State 0 (S_0):
if **C**: go to 1 (S_C)
else: stay here

State 1 (S_C):
if **C**: stay here
else if **H**: go to 2 (S_{CH})
else if **Y**: go to 3 (S_{CY})
else: return to 0 (S_0)

State 2 (S_{CH}):
if **H**: output **CHH** & return to 0 (S_0)
else if **Y**: output **CHY**, return to 0 (S_0)
else if **C**: return to 1 (S_C)
else: return to 0 (S_0)



anything but **CHY**

anything but **CH**

	INPUT	OUTPUT
¶1:	~CHY	none
	H	CHH
	Y	CHY
¶2:	~CH	none
	H	CYH

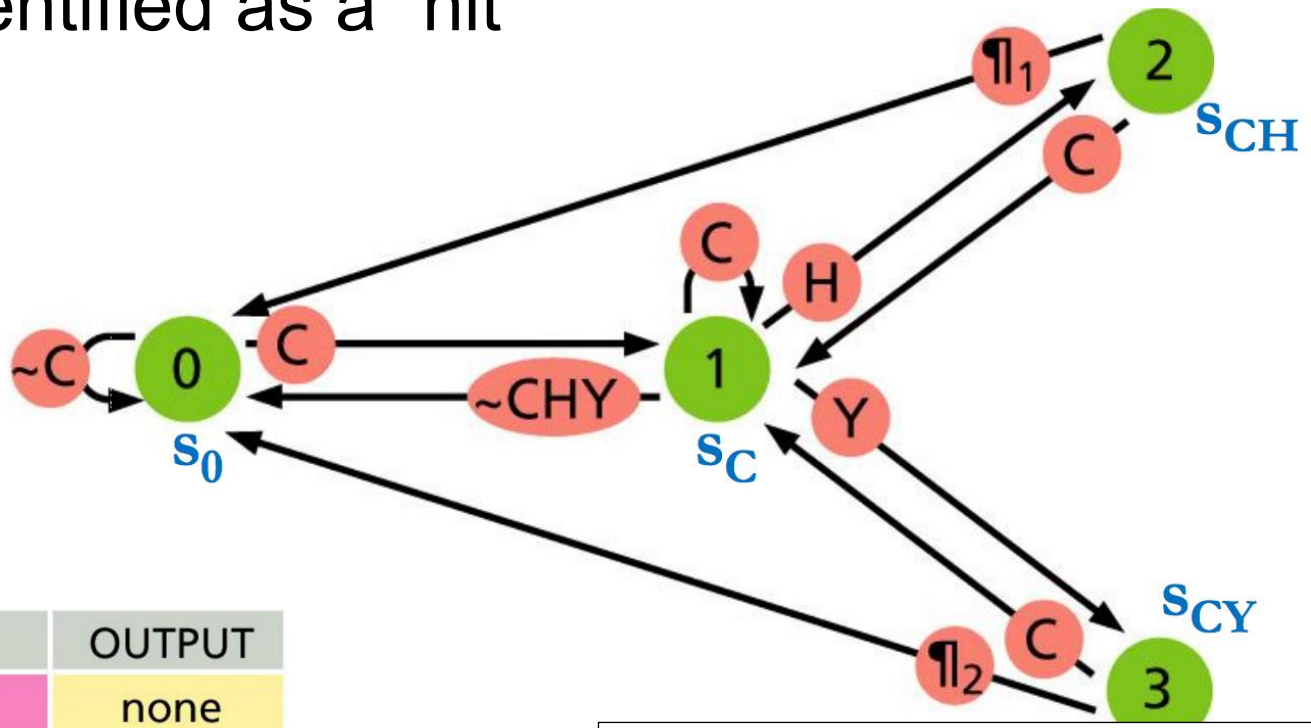
State 3 (S_{CY}):
if **H**: output **CYH** & return to 0 (S_0)
else if **C**: return to 1 (S_C)
else: return to 0 (S_0)

FSM

Example

e.g. a (subject) sequence from the database

If the input is **CHCYHC**, the states visited are **0–1–2–1–3–0–1**, with CYH identified as a “hit”



	INPUT	OUTPUT
¶1:	~CHY	none
	H	CHH
	Y	CHY
¶2:	~CH	none
	H	CYH

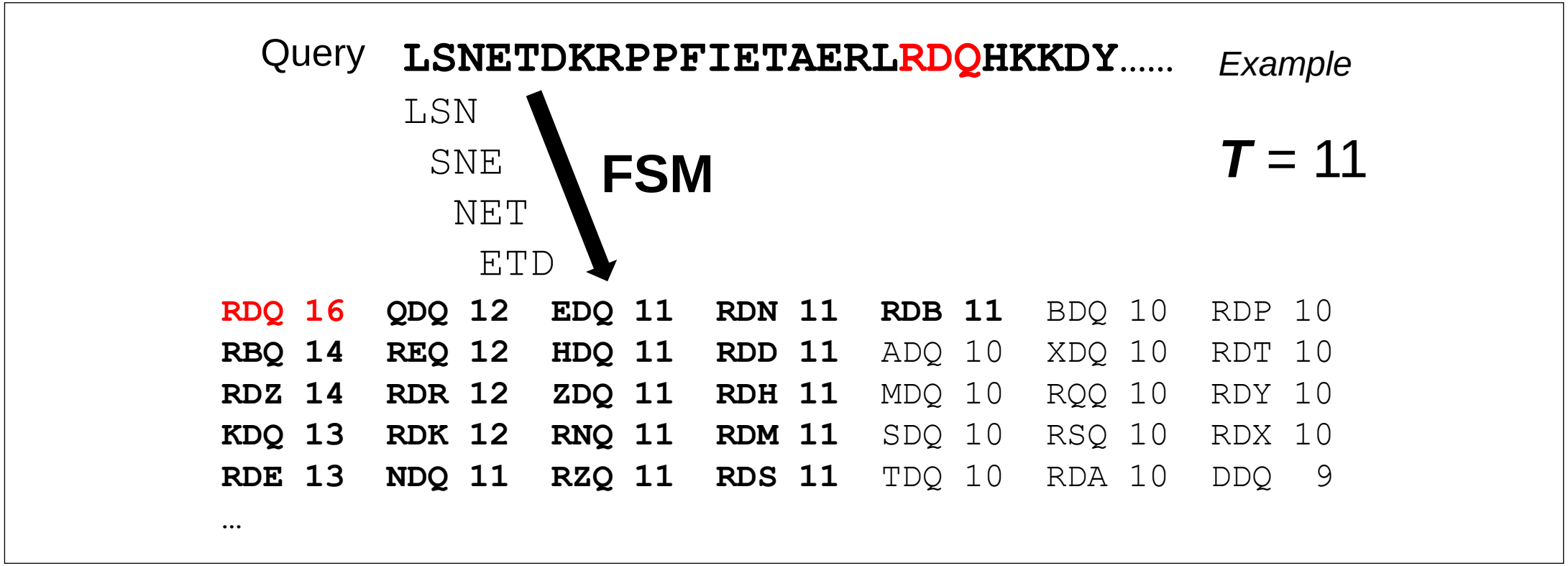
anything but **CHY**

anything but **CH**

State 3 (S_{CY}):
if **H**: output **CYH** & return to 0 (S_0)
else if **C**: return to 1 (S_C)
else: return to 0 (S_0)

BLAST algorithm

Phase 2: *identify k-mers in query with scores above threshold T for seeding*



This list represents *k*-mers (in any target/database sequence) that would give a score of ***T*** or higher when aligned with the query sequence; they are used for **seeding**

BLAST algorithm

Phase 3: *identify matches of these k-mers in each database sequence*

For each target sequence in the database:

- scan for hits with the compiled list of *k*-mers in Phase 2
- extend the hits to form **high-scoring segment pairs (HSPs)**
- Extension proceeds in **both** directions from the *k*-mer until the score drops by more than ***X*** relative to the current best score
- identify the highest scoring pair, i.e. maximal segment pair (MSP); if needed, combine two or more HSPs into a longer alignment
- final alignment score, ***S***

BLAST algorithm

Query: GSVEDTTGSQSLAALLNKCKT**PQG**QRLVNQWIKQPLMDKNRIEERLNLVEAFVEDAELRQTLQEDL

PQG 18
PEG 15
PRG 14
PKG 14
PNG 13
PDG 13
PHG 13
PMG 13
PSG 13
PQA 12
PQN 12
etc.

T = 13

extension proceeds in both directions

```
Query:   325 SLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEA 365
             +LA++L+   TP G R++ +W+  P+ D   + ER   + A
Sbjct:   290 TLASVLDCTVTPMGSRMLKRWLHMPVRDTRVLLERQQTIGA 330
```

High-scoring segment pair (HSP)

Example

BLAST algorithm

Phase 4: *evaluate the statistical significance of the alignments*

Expect Value (*E*-value)

- the number of alignments with a score **equal to or greater than** the observed score, that would be expected by chance alone in searching a database of n sequences
- *E*-value = 1: we would expect to see 1 match with the observed score or higher simply by chance
- the smaller an *E*-value, the more “significant” the match is
- dependent on the size of the database