Strings, Sequences, and Dynamic Programming

Algorithmic Thinking

Luay Nakhleh

Department of Computer Science

Rice University

Strings

- Strings play an important role in computer science.
- Strings are defined over a given alphabet Σ.
- For example, every "English string" is defined over the alphabet Σ={a,..,z,A,..,Z}.
- DNA strings are defined over the alphabet Σ={A,C,T,G}.
- RNA strings are defined over the alphabet Σ={A,C,U,G}.
- Binary strings are defined over the alphabet Σ={0,1}.
- We denote by Σ^* the set of all strings defined over the alphabet Σ. This set includes a special string, ε, which is the empty string (the string that contains no symbols).

Strings and Sequences

- Both strings and sequences are ordered lists of letters over an alphabet.
- Their main difference is best explained in terms of the difference between substrings and subsequences:
- u is a substring of v if there exists $x,y \in \Sigma^*$ such that $xuy=v$.
- u is a subsequence of v if u can be obtained by removing some letters from v.
- E.g., ACT is a substring of string ACTTT, but not a substring of ATCT.
- E.g., ACT is a subsequence of ATCT, but not a subsequence of TCA.

RNA Secondary Structure

- An RNA molecule is a sequence of n symbols (bases) drawn from the alphabet Σ={A,C,U,G}.
- Let $B=b_1b_2...b_n$ be an RNA molecule, where each $b_i \in \Sigma$.
- The RNA molecule forms a secondary structure based on a set of rules.

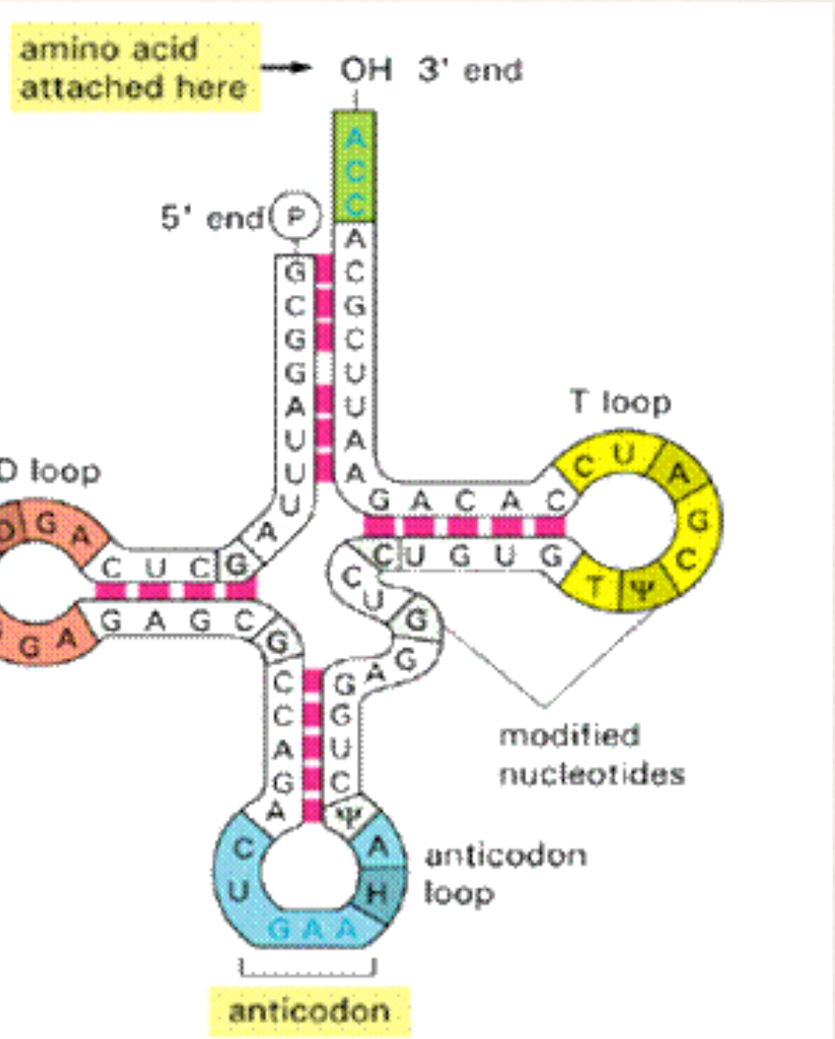


Figure 6-8. The "cloverleaf" structure of tRNA.
Molecular Biology of the Cell, 3rd Ed. Part II. Molecular Genetics
Chapter 6. Basic Genetic Mechanisms, RNA and Protein Synthesis

RNA Secondary Structure: Feasibility

- A secondary structure on B is a set of pairs S={(i,j)}, where i,j∈{1,2,...,n}, that satisfies the following conditions:

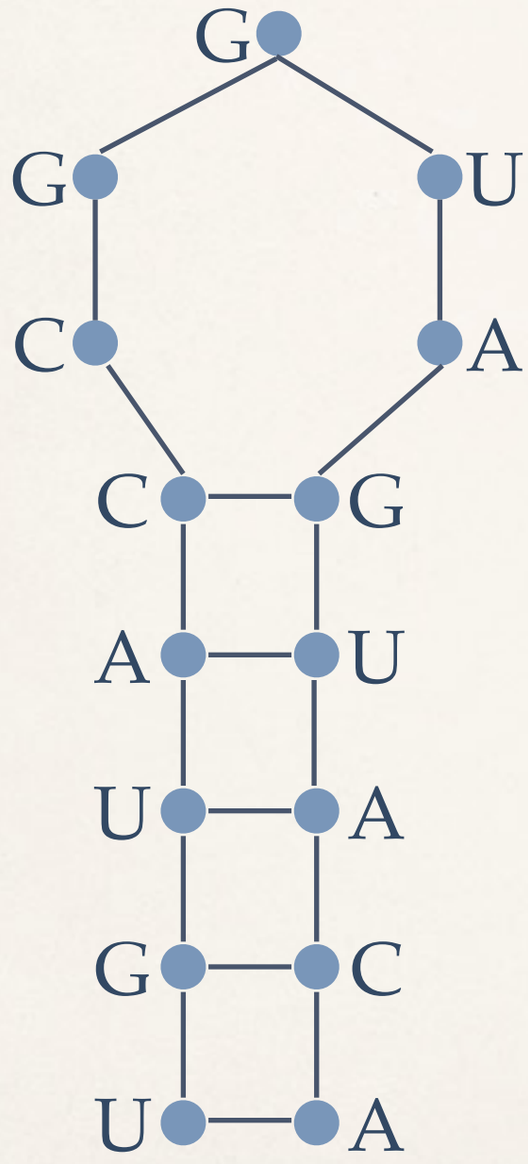
- The ends of each pair in S are separated by at least four intervening bases; that is, if (i,j)∈S, then i<j-4 (the "no sharp turns" condition).

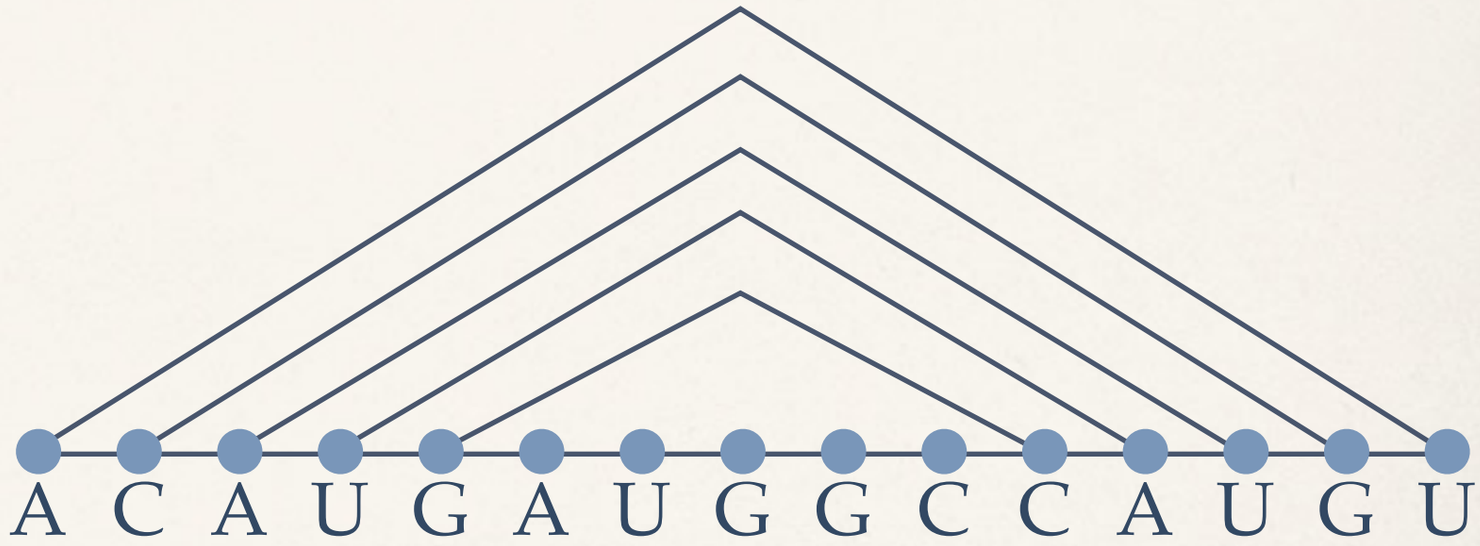
- The elements of any pair in S consist of either {A,U} or {C,G}.

- S is a matching: no base appears in more than one pair.

- If (i,j) and (k,l) are two pairs in S, then we cannot have i<k<j<l (the "noncrossing" condition).

RNA Secondary Structure





RNA Secondary Structure: Optimality

- Clearly, many RNA secondary structures may exist for a given RNA molecule.
- Out of all the feasible ones, which are the ones that are likely to arise under physiological conditions?
- A standard hypothesis is that an RNA molecule will form the secondary structure with the optimum total free energy.
- The correct model for the free energy of a secondary structure is a subject of much debate.
- A first approximation here is to assume that the free energy is proportional simply to the number of base pairs it contains.

RNA Secondary Structure: Solution = Feasibility + Optimality

- The RNA Secondary Structure Prediction Problem can now be defined as:

  - Input: RNA molecule $B=b_1b_2...b_n$

  - Output: A secondary structure S with the maximum possible number of base pairs.

How do we solve this problem exactly?

RNA Secondary Structure Prediction

- Denote by OPT(i,j) the maximum number of base pairs in a secondary structure on $b_i b_{i+1} \ldots b_j$.

- By the no-sharp-turns condition, we know that OPT(i,j)=0 whenever $i \geq j-4$.

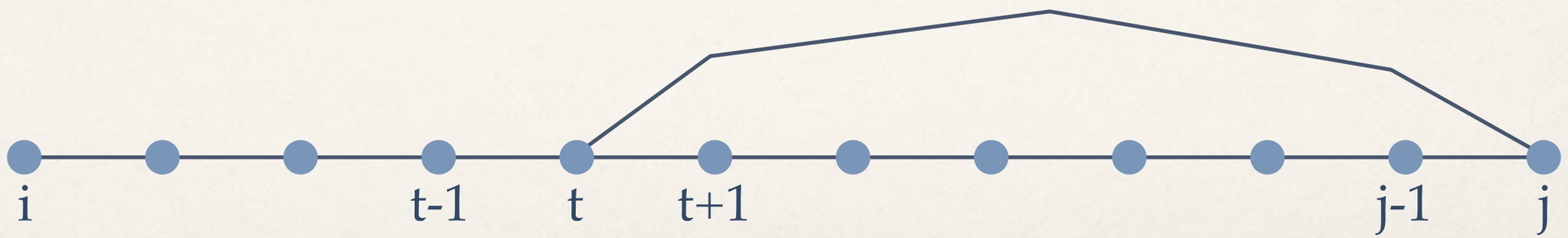
- Further, we know OPT(1,n) is the solution we're looking for.

- Let's now reason about OPT(i,j).

RNA Secondary Structure Prediction

- In the optimal secondary structure on $b_i b_{i+1} \ldots b_j$, we have one of two cases:
  - either j is not involved in a pair; or,
  - j pairs with t, for some $t<j-4$.

- In the first case, we have $OPT(i,j)=OPT(i,j-1)$.
- In the second case, $OPT(i,j)=1+OPT(i,t-1)+OPT(t+1,j-1)$.

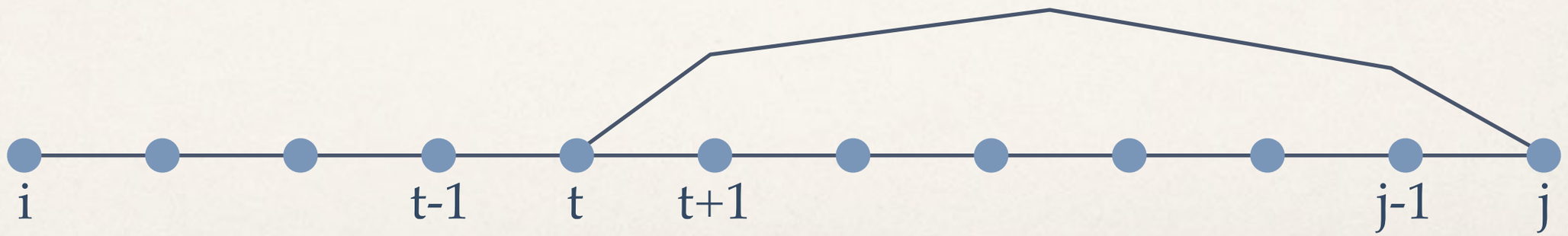


RNA Secondary Structure Prediction

- In the optimal secondary structure on $b_i b_{i+1} \ldots b_j$, we have one of two cases:
  - either j is not involved in a pair; or,
  - j pairs with t, for some $t<j-4$.

- In the first case, we have $OPT(i,j)=OPT(i,j-1)$.
- In the second case, $OPT(i,j)=1+OPT(i,t-1)+OPT(t+1,j-1)$.

Enabled by the noncrossing condition



RNA Secondary Structure Prediction

- There may be more than a single t value for which b_j and b_t can form a pair.
- Therefore, we have the following relationship:

(**) OPT(i,j) = max(OPT(i,j-1) , max(1+OPT(i,t-1)+OPT(t+1,j-1)))

the max is taken over t such that b_j and b_t are an allowable base pair.

RNA Secondary Structure Prediction

Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
    For i←1,2,...,n-k
        j←i+k;
        Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)

RNA Secondary Structure Prediction

Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
  For i←1,2,...,n-k
    j←i+k;
    Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)

RNA sequence ACCGGUAGU

RNA Secondary Structure Prediction

Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
  For i←1,2,...,n-k
    j←i+k;
    Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)

RNA sequence ACCGGUAGU

i=	j= 6	7	8	9
4	0	0	0	
3	0	0		
2	0			
1				

Initial values

RNA Secondary Structure Prediction

Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
  For i←1,2,...,n-k
    j←i+k;
    Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)

RNA sequence ACCGGUAGU

4	0	0	0	
3	0	0		
2	0			
i= 1				
j=	6	7	8	9

Initial values

4	0	0	0	0
3	0	0	1	
2	0	0		
i= 1	1			
j=	6	7	8	9

k=5

RNA Secondary Structure Prediction

Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
  For i←1,2,...,n-k
    j←i+k;
    Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)

RNA sequence ACCGGUAGU

i=	4	0	0	0	
	3	0	0		
	2	0			
	1				
j=		6	7	8	9

Initial values

i=	4	0	0	0	0
	3	0	0	1	
	2	0	0		
	1	1			
j=		6	7	8	9

k=5

i=	4	0	0	0	0
	3	0	0	1	1
	2	0	0	1	
	1	1	1		
j=		6	7	8	9

k=6

RNA Secondary Structure Prediction

```
Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
    For i←1,2,...,n-k
        j←i+k;
        Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)
```

RNA sequence ACCGGUAGU

i \ j	6	7	8	9
4	0	0	0	
3	0	0		
2	0			
1				

Initial values

i \ j	6	7	8	9
4	0	0	0	0
3	0	0	1	
2	0	0		
1	1			

k=5

i \ j	6	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	
1	1	1		

k=6

i \ j	6	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
1	1	1	1	

k=7

RNA Secondary Structure Prediction

```
Initialize OPT(i,j)←0 whenever i≥j-4;
For k←5,6,...,n-1
    For i←1,2,...,n-k
        j←i+k;
        Compute OPT(i,j) using the relationship (**);
Return OPT(1,n)
```

RNA sequence ACCGGUAGU

i=	j=6	7	8	9
4	0	0	0	
3	0	0		
2	0			
1				

Initial values

i=	j=6	7	8	9
4	0	0	0	0
3	0	0	1	
2	0	0		
1	1			

k=5

i=	j=6	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	
1	1	1		

k=6

i=	j=6	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
1	1	1	1	

k=7

i=	j=6	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
1	1	1	1	2

k=8

RNA Secondary Structure Prediction

- How do we get the actual secondary structure from the solution?
- What is the running time of the algorithm?

Dynamic Programming

- The algorithm that we have just seen for solving the RNA Secondary Structure Problem uses the Dynamic Programming (DP) algorithmic technique.

- Dynamic programming is a technique for solving problems with overlapping subproblems.

- Typically, these subproblems arise from a recurrence relating a solution to a given problem with solutions to its smaller subproblems of the same type.

- Rather than solving overlapping subproblems again and again, dynamic programming suggests solving each of the smaller subproblems only once and recording the results in a table from which we can then obtain a solution to the original problem.

Dynamic Programming

- To set about developing an algorithm based on dynamic programming, one needs a collection of subproblems derived from the original problem that satisfies a few basic properties:

- The solution to the original problem can be easily computed from the solutions to the subproblems (for example, the original problem may actually be one of the subproblems).

- There is a natural ordering on subproblems from “smallest” to “largest,” together with an easy-to-compute recurrence that allows one to determine the solution to a subproblem from the solutions to some number of smaller subproblems.

Illustrating the Properties of DP Algorithms: The RNA Secondary Structure Prediction Problem

the max is taken over t such that b_j and b_t are an allowable base pair.

(**) OPT(i,j) = max(OPT(i,j-1) , max(1+OPT(i,t-1)+OPT(t+1,j-1)))

- Solution from solutions to subproblems?
- Natural ordering of subproblems?

Polynomial DP Algorithms

- For the DP algorithm to be polynomial, the number of subproblems that need to be solved must be polynomial.

Evolution and Sequence Alignment

Life through Evolution

- All living organisms are related to each other through evolution.
- This means: any pair of organisms, no matter how different, have a common ancestor sometime in the past, from which they evolved.
- Evolution involves
  - inheritance: passing of characteristics from parent to offspring
  - variation: differentiation between parent and offspring
  - (and other processes, such as selection,...)

Sequence Variations Due to Mutations

- Mutations and selection over millions of years can result in considerable divergence between present-day sequences derived from the same ancestral sequence.

- The base pair composition of the sequences can change due to point mutation (substitutions), and the sequence lengths can vary due to insertions/deletions.

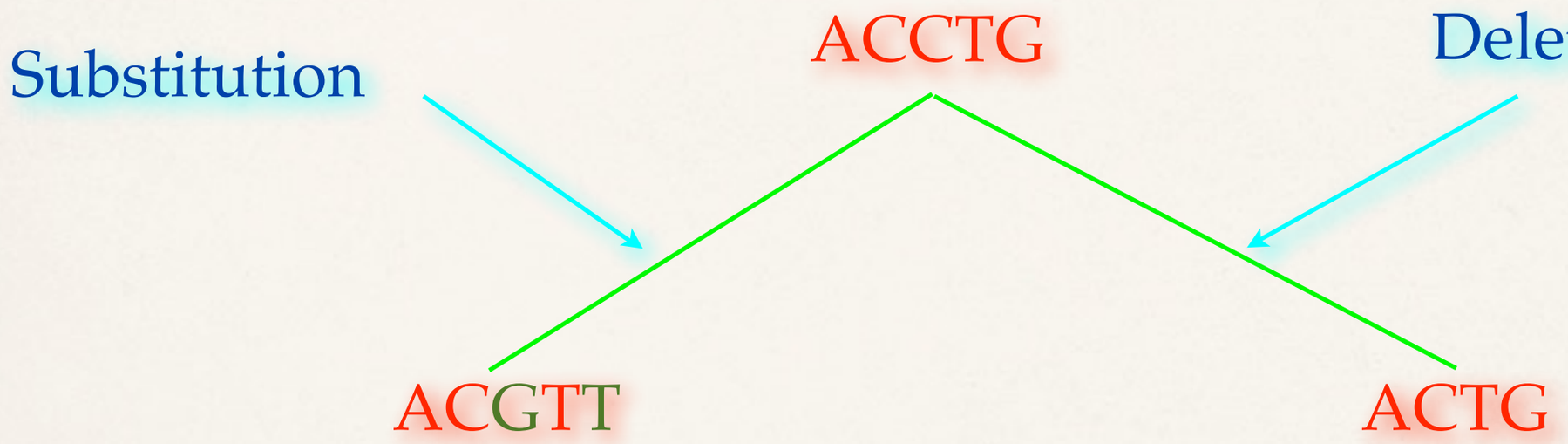
Sequence Evolution

Sequence Evolution

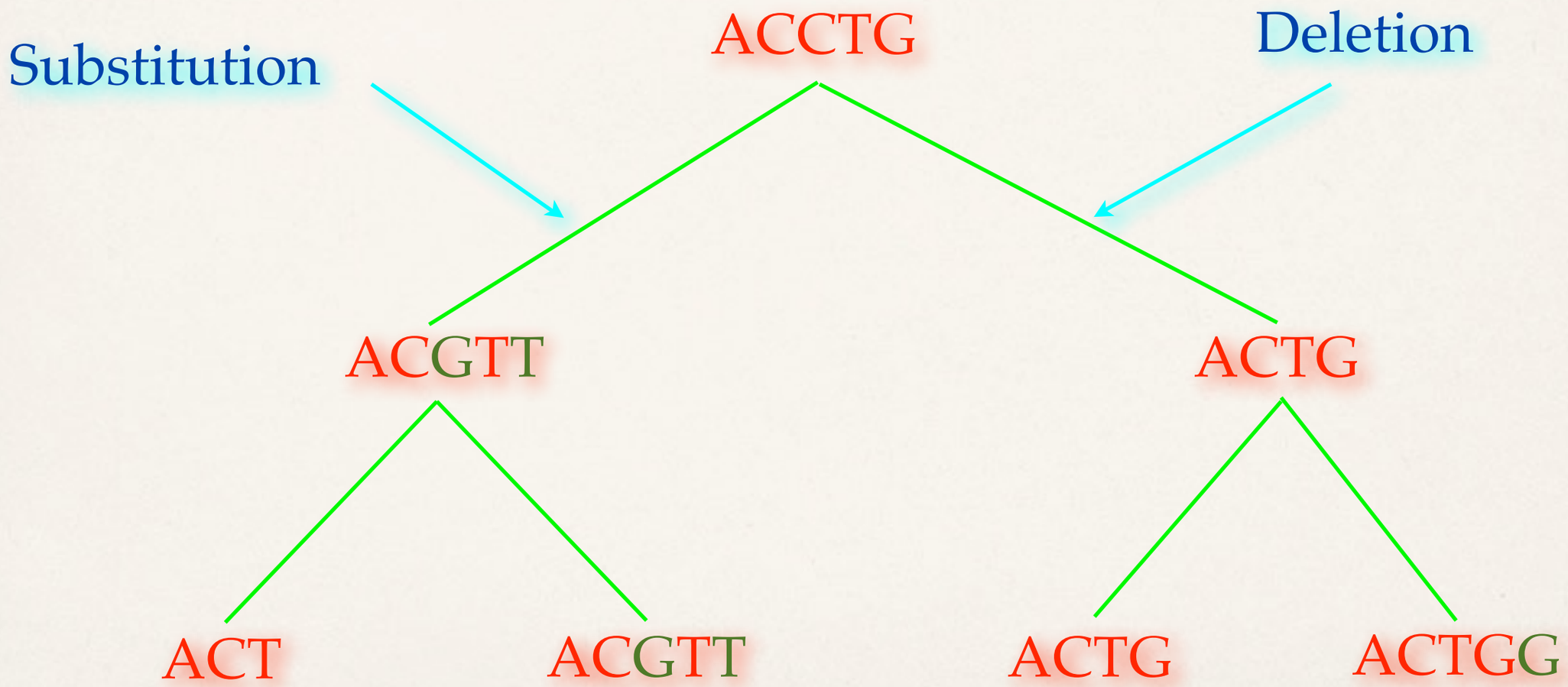
ACCTG

Sequence Evolution

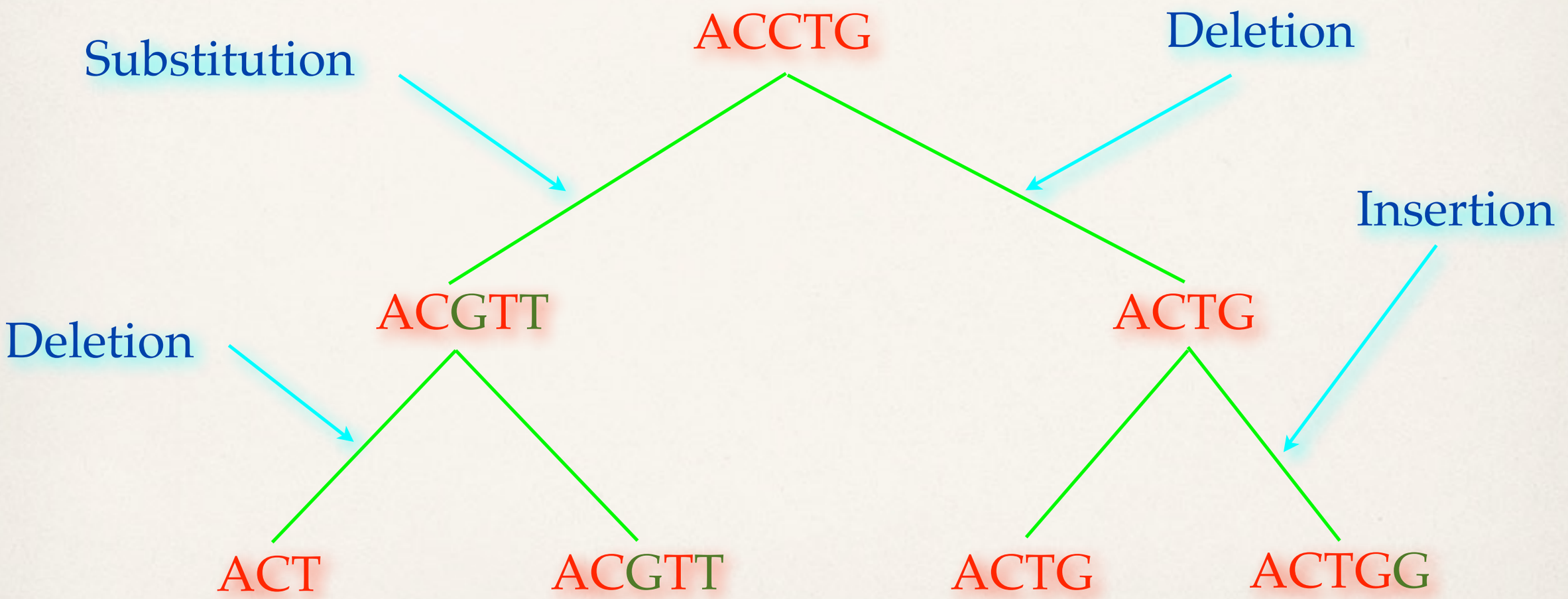
Sequence Evolution



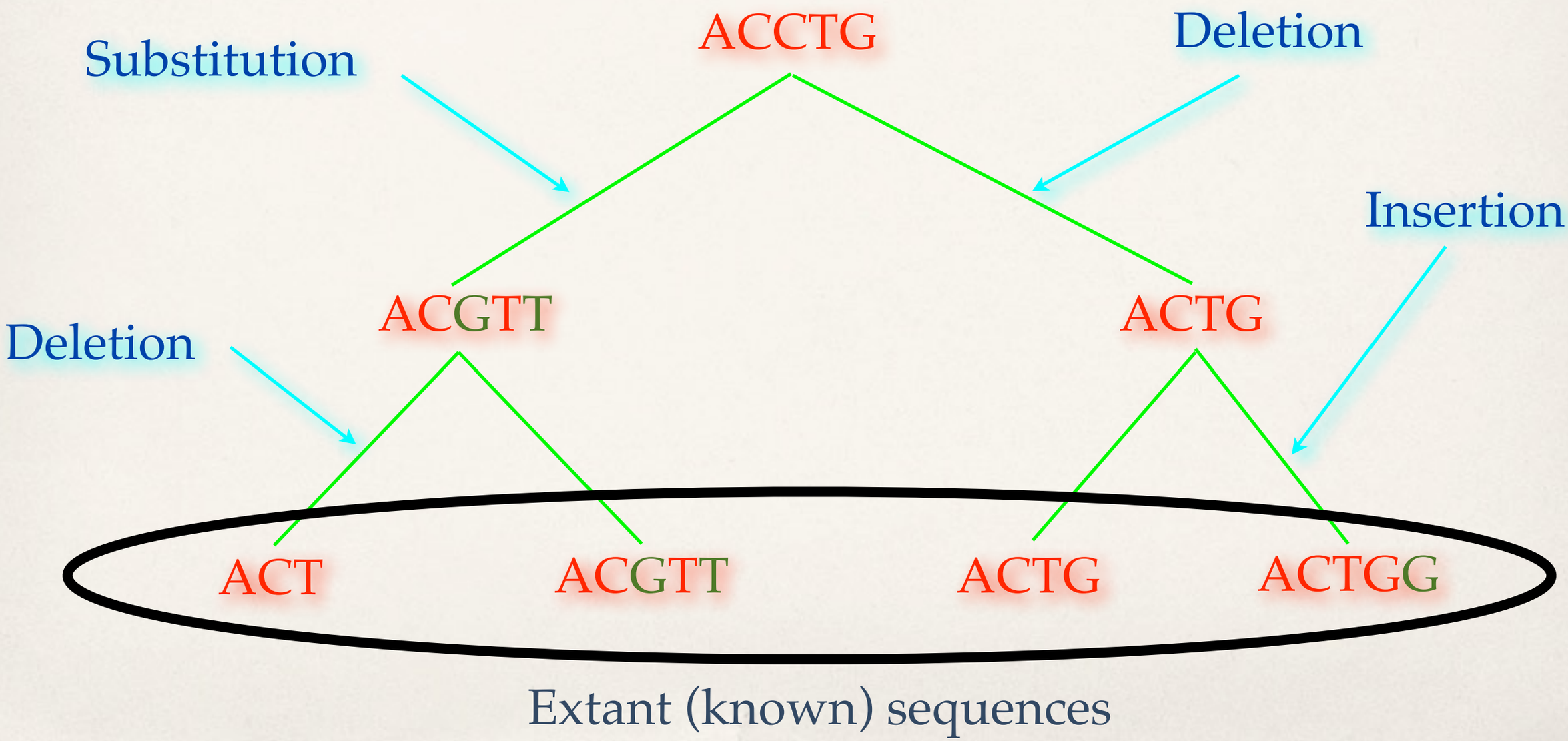
Sequence Evolution



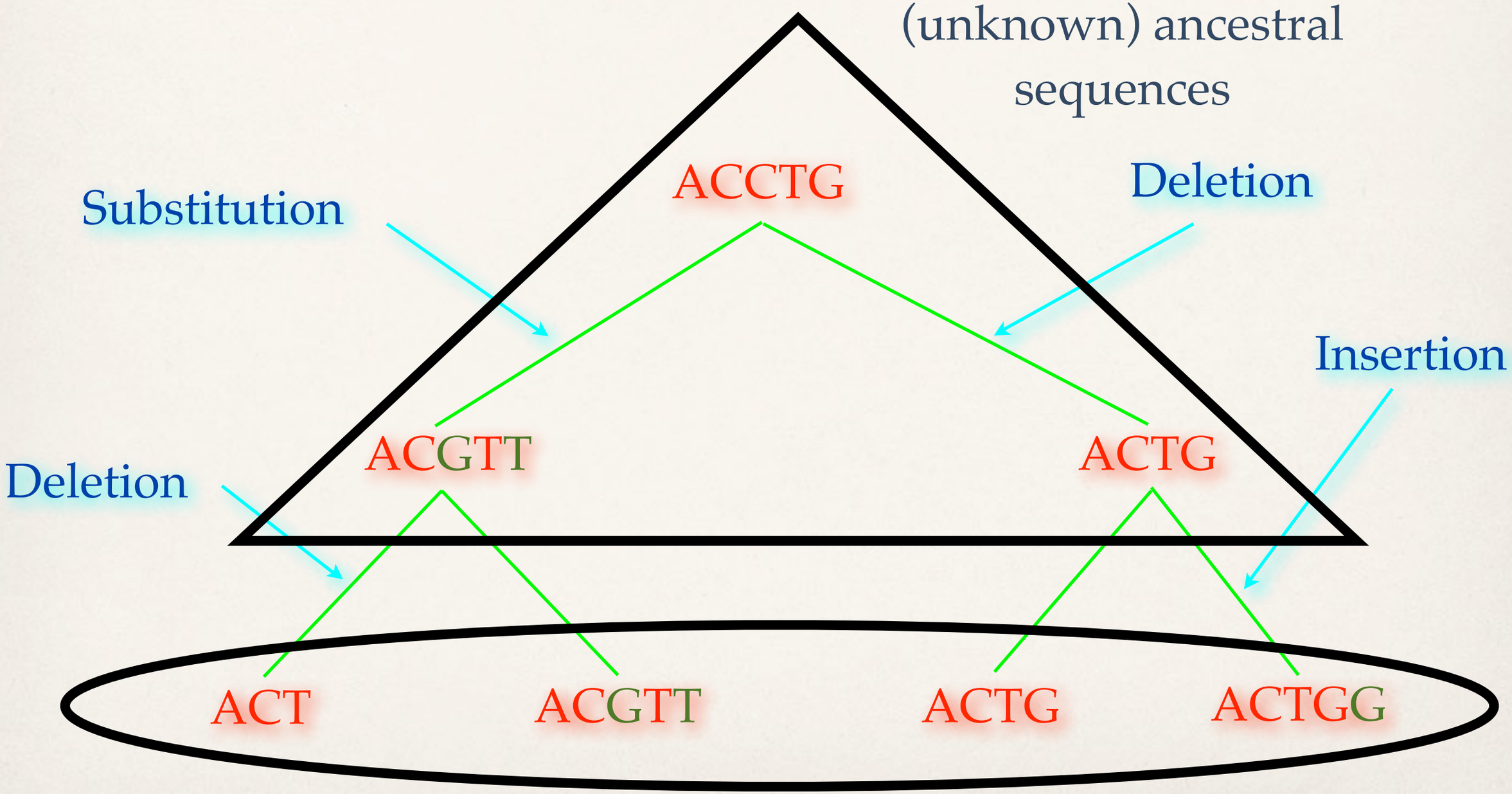
Sequence Evolution



Sequence Evolution



Sequence Evolution



Sequence Evolution

- In biology, we have access to the extant (known) sequences, but in most cases no knowledge of the ancestral sequences.

- Therefore, a central task in biology is to identify similarities and differences between extant sequences in an attempt to map the evolutionary past.

- Using the example of the previous slide, we are interested in finding the similarities, for example, between the two sequences ACT and ACGTT.

- As sequences change in length and content throughout evolution, we are often interested in regions of high similarities between the two sequences.

- We can be “very strict” (similarity=identity) or “less strict” (similarity includes matches, mismatches, and gaps).

The Longest Common Subsequence (LCS) Problem

- In the case of the LCS problem, we seek the longest sequence that is a subsequence of two input sequences X and Y.
- For example, if X=ACT and Y=ACGTT,
  - AC is a subsequence of X as well as of Y.
  - CT is a subsequence of X as well as of Y.
  - However, ACT is the longest sequence that is a subsequence of X and at the same time a subsequence of Y.

The Longest Common Subsequence (LCS) Problem

- In the case of the LCS problem, we seek the longest sequence that is a subsequence of two input sequences X and Y.

- For example, if X=ACT and Y=ACGTT,
  - AC is a subsequence of X as well as of Y.
  - CT is a subsequence of X as well as of Y.
  - However, ACT is the longest sequence that is a subsequence of X and at the same time a subsequence of Y.

X=ACT X=ACGTT

The Longest Common Subsequence (LCS) Problem

- Give a brute-force algorithm for solving the LCS Problem.
- Do you think the algorithm is efficient?
- Now, reason about the problem "recursively": Let X=X'a and Y=Y'b, where a and b are single letters, and X' and Y' are strings (in other words, X ends with the letter a, and Y ends with the letter b).
- There are two cases:
  - a=b: Are they part of an LCS solution? If not, how do we proceed?
  - a≠b: Are they part of an LCS solution? If not, how do we proceed?

The Longest Common Subsequence (LCS) Problem

- The recursive reasoning naturally gives rise to an algorithm for solving the LCS problem efficiently, by making use of solutions to sub-problems.

- While computationally the problem is "taken care of," biologically it is expected that the more divergent the two sequences X and Y are, the shorter the subsequences that are common to both become.

- Therefore, in most cases, it is necessary that we relax the "identity constraint," and instead seek similarities across the two sequences that include matches (letters that are identical in both sequences), mismatches (letters are that are not identical in both, but are accepted as pairs), and gaps (letters that are present in one sequence but missing from the other).

- This is known as the sequence alignment problem.

Relaxing the Identity Constraint:

Sequence Alignment (matches, mismatches, and gaps)

T H A T S E Q U E N C E

T H I S S E Q U E N C E

Relaxing the Identity Constraint:

Sequence Alignment (matches, mismatches, and gaps)

T H A T S E Q U E N C E

T H I S S E Q U E N C E

LCS Solution ⟶ THSEQUENCE

Relaxing the Identity Constraint:
Sequence Alignment (matches, mismatches, and gaps)

T H A T S E Q U E N C E

T H I S S E Q U E N C E

LCS Solution ⟶ THSEQUENCE

T H A T S E Q U E N C E

T H I S S E Q U E N C E

Relaxing the Identity Constraint:
Sequence Alignment (matches, mismatches, and gaps)

T H A T S E Q U E N C E

T H I S S E Q U E N C E

LCS Solution ⟶ THSEQUENCE

T H A T S E Q U E N C E

T H I S S E Q U E N C E

Alignment with Mismatches

Relaxing the Identity Constraint:

Sequence Alignment (matches, mismatches, and gaps)

T H A T S E Q U E N C E

T H I S S E Q U E N C E

LCS Solution ⟶ THSEQUENCE

T H A T S E Q U E N C E

T H I S S E Q U E N C E

Alignment with Mismatches

T H I S I S A – S E Q U E N C E

T H – – – – A T S E Q U E N C E

Relaxing the Identity Constraint:

Sequence Alignment (matches, mismatches, and gaps)

T H A T S E Q U E N C E

T H I S S E Q U E N C E

LCS Solution ⟶ THSEQUENCE

T H A T S E Q U E N C E

T H I S S E Q U E N C E

Alignment with Mismatches

T H I S I S A – S E Q U E N C E

T H – – – – A T S E Q U E N C E

Alignment with Gaps (indels: insertions / deletions)

Sequence Alignment

- As you can imagine, since we don't enforce identity, any way of "aligning" the two sequences X and Y so that their lengths are equal is a "candidate" for sequence alignment (just pad them with dashes so that their lengths are equal; don't align dash with dash, though).

- So, how do we choose the "best" alignment?

- We define a scoring matrix that gives a score to every pair of aligned letters, and a penalty to every column with a dash.

- The score of an alignment is then the sum of the scores and penalties assigned to each column in the alignment.

- The "best" alignment is one with the highest score.

Sequence Alignment

- Here's an example of a scoring matrix M, where M_{pq}=i indicates that if p and q are aligned with each other in the alignment, they contribute score i to the overall score of the alignment (and penalty i if either p or q is a dash).

- Consider the alphabet {A,C,T,G}.

- Here's an example of a scoring matrix M.

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	2	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/A

Sequence Alignment

X=ACC
Y=AGC

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	5	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/

Sequence Alignment

X=ACC
Y=AGC

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	5	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/

	Alignment 1	Alignment 2	Alignment 3	Alignment 4
X′=	A C C - -	A C - C	A C C - - -	A C C
Y′=	- - A G C	A - G C	- - - A G C	A G C

Sequence Alignment

X=ACC
Y=AGC

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	5	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/

	Alignment 1	Alignment 2	Alignment 3	Alignment 4
X′=	A C C - -	A C - C	A C C - - -	A C C
Y′=	- - A G C	A - G C	- - - A G C	A G C
column scores	-6 | -5 | 5 | -2 | -4	10 | -5 | -2 | 10	-6 | -5 | -5 | -4 | -2 | -4	10 | 5 | 10

Sequence Alignment

X=ACC
Y=AGC

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	5	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/

	Alignment 1	Alignment 2	Alignment 3	Alignment 4
X′=	A C C - -	A C - C	A C C - - -	A C C
Y′=	- - A G C	A - G C	- - - A G C	A G C
column scores	-6 | -5 | 5 | -2 | -4	10 | -5 | -2 | 10	-6 | -5 | -5 | -4 | -2 | -4	10 | 5 | 10
alignment score	**-12**	**13**	**-26**	**25**

Sequence Alignment

X=ACC
Y=AGC

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	5	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/

	Alignment 1	Alignment 2	Alignment 3	Alignment 4
X′=	A C C - -	A C - C	A C C - - -	A C C
Y′=	- - A G C	A - G C	- - - A G C	A G C
column scores	-6 | -5 | 5 | -2 | -4	10 | -5 | -2 | 10	-6 | -5 | -5 | -4 | -2 | -4	10 | 5 | 10
alignment score	**-12**	**13**	**-26**	**25**

The best among these four alignments

Sequence Alignment

X=ACC
Y=AGC

	A	C	T	G	-
A	10	5	7	3	-6
C	5	10	6	5	-5
T	5	1	15	1	-3
G	8	4	2	15	-1
-	-4	-4	-2	-2	N/

These are just 4 alignments.. there are many more possible ones!

	Alignment 1	Alignment 2	Alignment 3	Alignment 4
X′=	A C C - -	A C - C	A C C - - -	A C C
Y′=	- - A G C	A - G C	- - - A G C	A G C
column scores	-6 | -5 | 5 | -2 | -4	10 | -5 | -2 | 10	-6 | -5 | -5 | -4 | -2 | -4	10 | 5 | 10
alignment score	**-12**	**13**	**-26**	**25**

The best among these four alignments

Sequence Alignment

- Is a brute-force algorithm feasible for this problem?
- Can we come up with a better algorithm that is feasible for practical cases?

Sequence Alignment

- Is a brute-force algorithm feasible for this problem?
- Can we come up with a better algorithm that is feasible for practical cases?

Answer: Enjoy Module 4!