

# **4. Finding Regulatory Motifs in DNA Sequences (Chapter 4 and 12)**

# Outline

- Implanting Patterns in Random Text
- Gene Regulation
- Regulatory Motifs
- The Motif Finding Problem
- Brute Force Motif Finding
- The Median String Problem
- Search Trees
- Branch-and-Bound Motif Search
- Branch-and-Bound Median String Search
- Consensus and Pattern Branching: Greedy Motif Search
- PMS: Exhaustive Motif Search

# Identifying Motifs

Every gene contains a regulatory region (RR) typically stretching 100-1000 bp upstream of the transcriptional start site

Genes are turned on or off by regulatory proteins

- These proteins bind to upstream regulatory regions of genes to either attract or block an RNA polymerase
- Regulatory protein **transcription factor (TF)** binds to a short DNA sequence called a **motif or (TFBS: transcription factor binding site)**
- So finding the same motif in multiple genes' regulatory regions suggests a regulatory relationship amongst those genes

# Identifying Motifs: Complications

- We do not know the motif sequence
- We do not know where it is located relative to the genes start
- Motifs can differ slightly from one gene to the next
- How to discern it from “random” motifs?

# Transcription Factor Binding Sites (TFBS)

- A TFBS can be located anywhere within the **Regulatory Region**.
- TFBS may vary slightly across different regulatory regions since non-essential bases could mutate

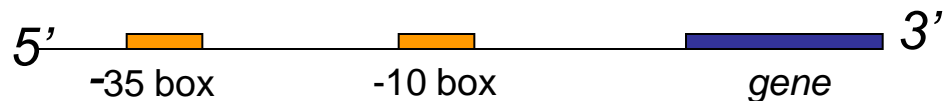
# Transcription for E.Coli

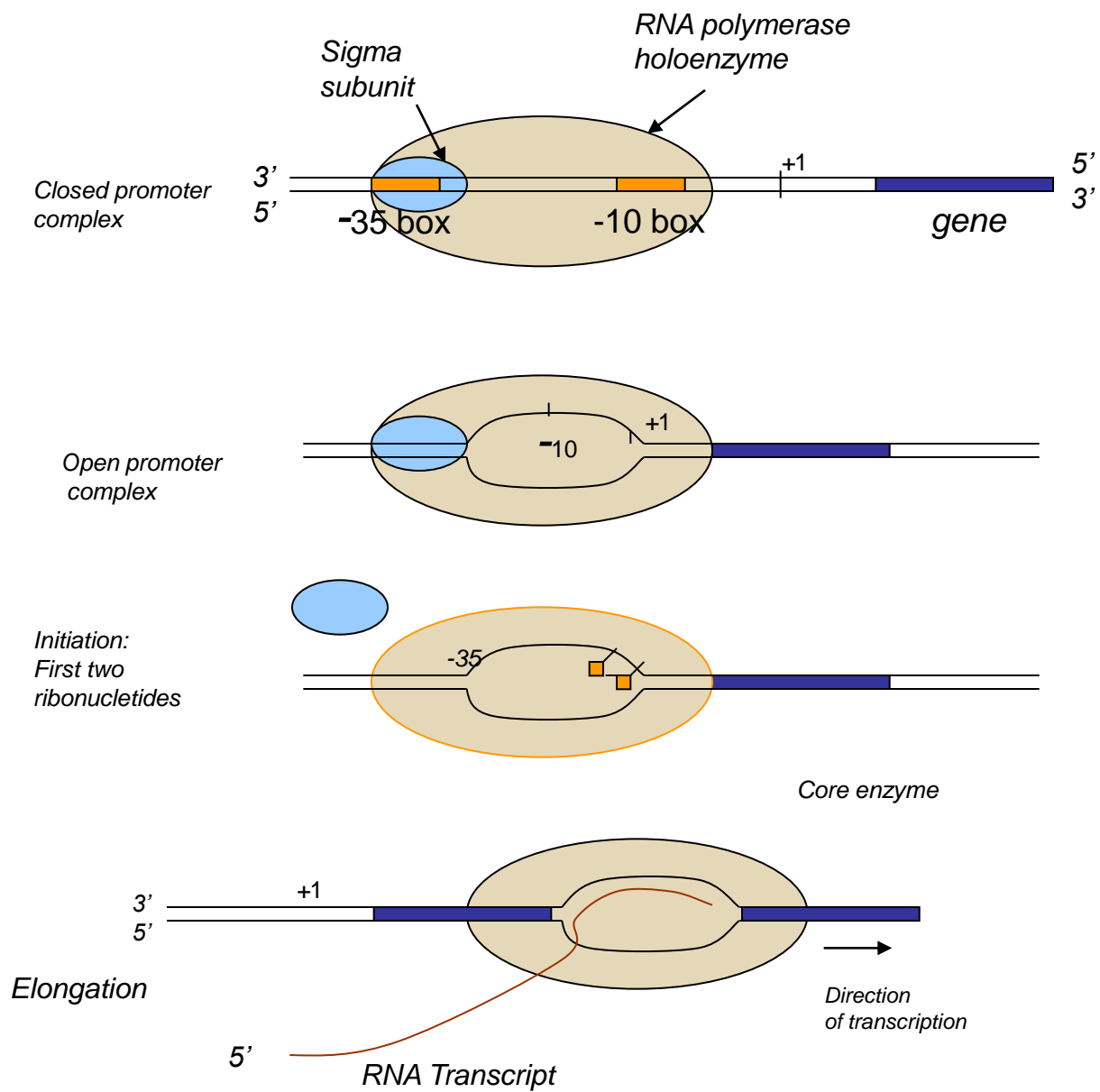
The transcription process in reality even for a simple E.Coli is much more complex than what we have described. The process is divided into three phases: **initiation, elongation and termination.**

**Initiation:** The RNA polymerase initiates the operation and it must transcribe not any arbitrary part of DNA but only the gene. For this the polymerase first 'bind' to a location **upstream** of the gene. This site is called a **promoter** sequence. The promoter is a short DNA sequence which can be bind to the polymerase. In E.Coli, the promoter sequence consists of two distinct sequences at a distance -10 and -35 upstream from the position at which transcription starts. The actual sequences may vary from gene to gene but they are related to the following two **consensus sequences** both located in the non-template strand:

**-35 box 5'-TTGACA-3'**

**-10 box 5'-TATAAT-3'**





# Steps of RNA Transcription

The sigma subunit within the polymerase recognizes the promoter sequence and a **closed promoter complex** is formed. The enzyme covers about 60 bp of the double helix. In the next phase, the double helix starts 'melting' at -10 box unwinding the DNA into single strands in the region under the core enzyme. The -10 box consists of entirely AT pairs which have only two hydrogen bonds for each bp. This makes it easier to melt to take place compared to the situation with CG base pairs which have 3 hydrogen bonds. This configuration is called **open promoter complex**, the sigma subunit ejects out of the holoenzyme converting it to a core enzyme. At the same time, the first two ribonucleotides are sealed in the template strand at positions +1 and +2 with a phosphodiester bond.

In the next **elongation step**, the polymerase moves downstream with relative ease, unzipping the DNA molecule and attaching new ribonucleotides to the 3' end of the growing RNA. At the same time, the DNA behind it rebounds back to its double helix structure. The open promoter is



like a bubble that propagate to 3' direction always maintaining its size between 12 to 17 DNA. Also, the rate of propagation is not constant, it may slow down, pause, reaccelerate or even go backwards destroying the ribonucleotides. The RNA itself is synthesized in the 5'-3' direction. The length of the actual transcript is longer than the length of the gene because the +1 position is about 20 to 600 nucleotide upstream from the beginning of the gene. This part of the RNA transcript is called a **leader segment**. Similarly, the transcription extends beyond the gene creating a similar **trailer segment**.

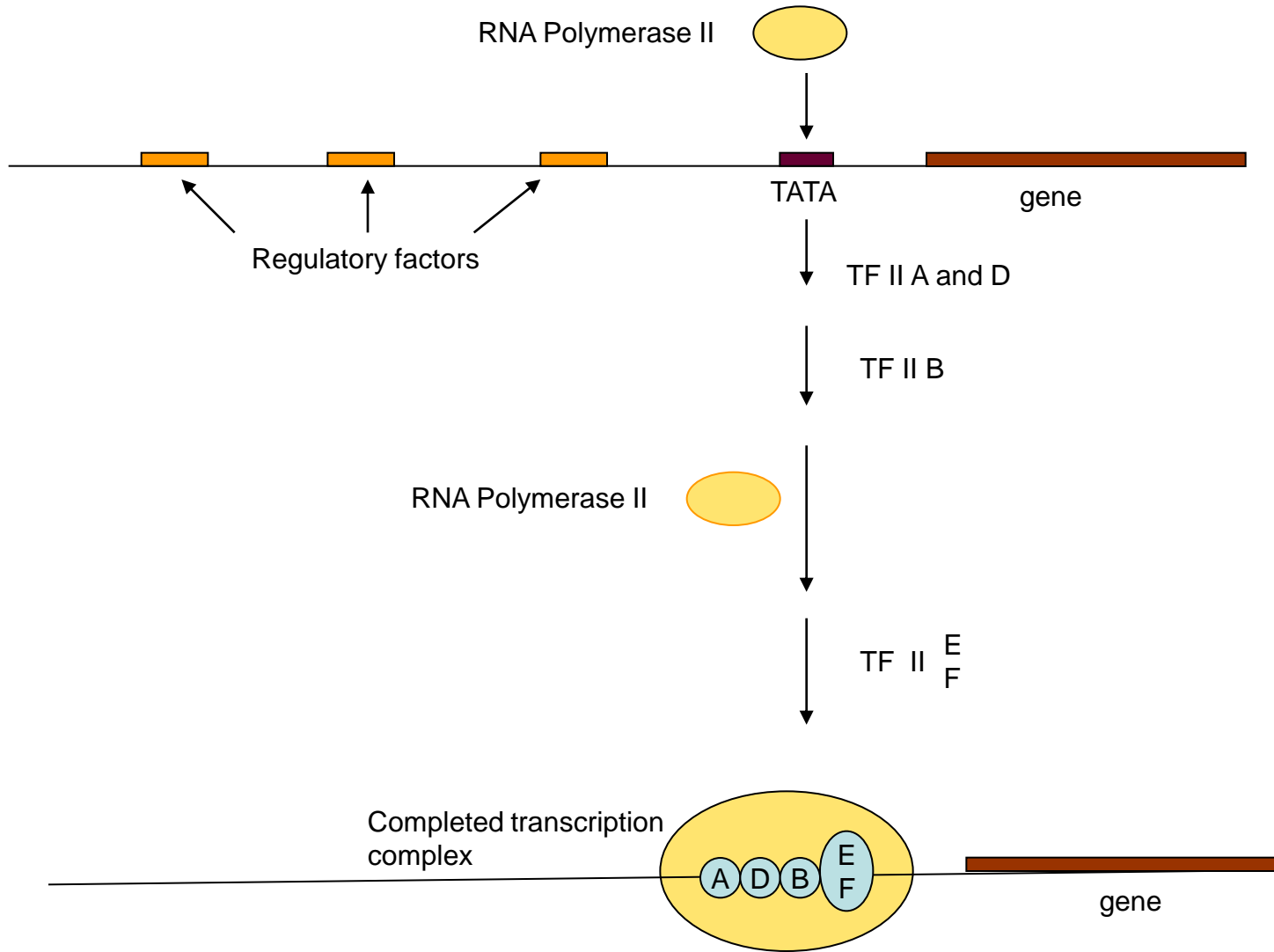
The **termination** of RNA transcription is signalled by the presence of a **complementary palindrome**. ( A palindrome reads the same sequence in both forward and backward direction viz ATAGCGATA )

# Transcription in Eukaryotes

The transcription in Eukaryotes is similar to that in E.Coli but is much more complex. The RNA polymerase has an **attachment site** rather than a promoter sequence plus other promoter sequences distributed over several hundred base pairs all upstream from the genes. These promoters regulate the gene expression by turning on or off the transcription process. Understanding these regulatory processes is by itself a whole new research field.

The attachment site is referred to as **-25 TATA box** (5'-TATAAAT-3') and the RNA polymerase called **RNA Polymerase II**. The attachment is helped by a set of proteins called **transcription factors (TF II A, TF II B , D, E and F)** which ultimately makes the transcription complex ready to start the synthesis process of RNA. The next slide gives a schematic representation.

The exact details of the termination of the transcription process is not very well understood. The termination trail seem to be longer ( about 1000 to 2000 bp downstream the gene. The exact termination point is still a matter of research.



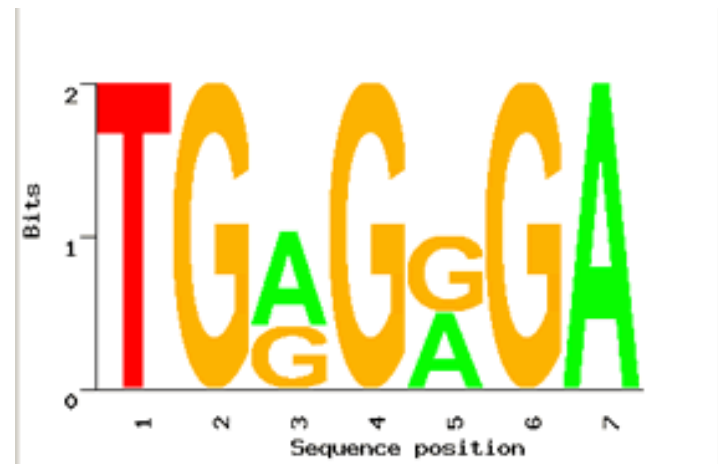
# Motifs and Transcriptional Start Sites



# Motif Logo

- Motifs can mutate on non important bases
- The five motifs in five different genes have mutations in position 3 and 5 and 5
- Representations called *motif logos* illustrate the conserved and variable regions of a motif

TGGGGGA  
TGAGAGA  
TGGGGGA  
TGAGAGA  
TGAGGGA



# Random Sample

atgaccgggatactgataccgtatTTGGCCTAGGCgtacacattagataaacgtatgaagtacgTTtagactcggcgccgccg  
accctatTTTTtgagcagatttagtgacctggaaaaaaaaatttgagtacaaaactTTTccgaatactgggcataaggtaca  
tgagtatccctgggatgactTTTgggaacactatagtgctctcccgattTTTgaatatgtaggatcattcgccaggggtccga  
gctgagaattggatgaccttgtaagtgtTTTccacgcaatcgcgaaaccaacgcggacccaaaggcaagaccgataaaggaga  
tccTTTTgCGGtaatgtgCCGGGaggctggTTacgtagggaagccctaacggacttaatggcccacttagtccacttatag  
gtcaatcatgttcttGTgaatggattTTTtaactgagggcatagaccgcttggcgcacccaaattcagtgtgggCGagCGcaa  
CGTTTTggcccttgTTtagaggccccgtactgatggaaactTTcaattatgagagagctaatttatCGcgtgcgtgttcat  
aacttgagttggTTTcgaaaatgctctggggcacatacaagaggagtcttccTTtatcagTTaatgctgtatgacactatgta  
ttggccattggctaaaagcccaacttgacaaatggaagatagaatccttgcaTTTcaacgtatgccgaaccgaaaggggaag  
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

# Implanting Motif AAAAAAAGGGGGGG

atgaccgggatactgatAAAAAAAGGGGGGGggcgctacacattagataaacgtatgaagtacgttagactcggcgccgccg  
accctatTTTTTgagcagatttagtgacctggaaaaaaatttgagtacaaaactTTTccgaataAAAAAAAGGGGGGGa  
tgagtatccctgggatgacttAAAAAAAGGGGGGGTgtctctcccgattTTTgaatatgtaggatcattcgccagggtccga  
gctgagaattggatgAAAAAAAGGGGGGGTccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga  
tccTTTTgcggaatgtgccgggaggctggttacgtagggaagccctaacggacttaataAAAAAAAGGGGGGGccttatag  
gtcaatcatgttcttTgtgaatggatttAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgTgggCGagcGcaa  
cggTTTTggcccttTgtagaggccccgtAAAAAAAGGGGGGGcaattatgagagagctaattctatcgcgTgcgtGttcat  
aacttgagttAAAAAAAGGGGGGGGctggggcacatacaagaggagtcttcccttatcagttaatgctgtatgacactatgta  
ttggccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAGGGGGGGgaccgaaaggaag  
ctggTgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAGGGGGGGa

# Where is the Implanted Motif?

atgaccgggatactgataaaaaaaagggggggggcggtacacattagataaacgtatgaagtacgtttagactcggcgccgccg  
accctatTTTTTgagcagatttagtgacctggaaaaaaatttgagtacaaaactTTTccgaataaaaaaaaaggggggga  
tgagtatccctgggatgacttaaaaaaaaggggggggtgctctcccgattTTTgaatatgtaggatcattcgccaggggtccga  
gctgagaattggatgaaaaaaaggggggggtccacgcaatcgcgaaaccaacgcggacccaaaggcaagaccgataaaggaga  
tccTTTTgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggggcttatag  
gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa  
cggTTTTggcccttgtttagaggccccgtaaaaaaaaggggggggcaattatgagagagctaatttatcgcggtgcgtgttcat  
aacttgagttaaaaaaaggggggggctggggcacatacaagaggagtcttcccttatcagttaatgctgtatgacactatgta  
ttggccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaaggggaag  
ctggtgagcaacgacagattcttacgtgcattagctcgcttccgggggatctaatagcacgaagcttaaaaaaaaggggggga



# Implanting Motif AAAAAAAGGGGGGGG with Four Mutations

atgaccgggatactgatAgAAgAAAGGttGGGggcggtacacattagataaacgtatgaagtacgtttagactcggcgccgccg  
accctatTTTTTgagcagatttagtgacctggaaaaaaatttgagtacaaaactTTTccgaataCAAtAAACGGcGGa  
tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgattTTTgaatatgtaggatcattcgccaggggtccga  
gctgagaattggatgCAAAAAAGGGattGtccacgcaatcgcgaaaccaacgcggaccCAAaggcaagaccgataaaggaga  
tccTTTTgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAAtAAAGGaaGGGcttatag  
gtcaatcatgttcttgtgaatggatttAAcAAAtAAGGGctGGgaccgcttggcgcacccaaattcagtggtgggCGagcgcaa  
cggTTTTggcccttgtttagaggccccgtAtAAAcAAGGaAGGGcCaattatgagagagctaattctatcgcggtgcgtgttcat  
aacttgagttAAAAAtAGGGaGccctggggcacatacaagaggagtcttcccttatcagttaatgctgtatgacactatgta  
ttggccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatacTAAAAAGGaGcGGaccgaaaggaag  
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa

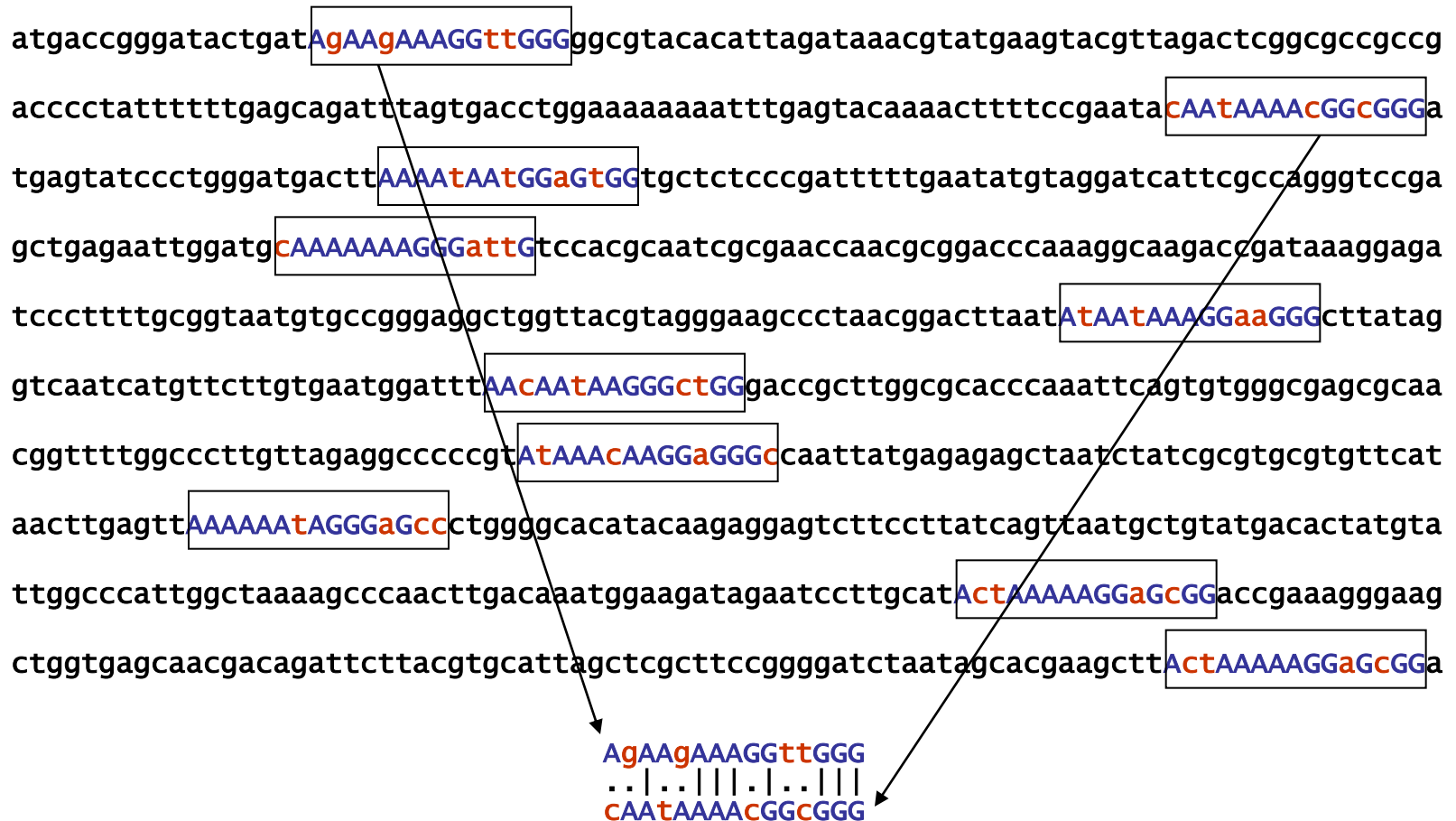
# Where is the Motif???

atgaccgggatactgatagaagaaagggttgggggcggtacacattagataaacgtatgaagtacgtttagactcggcgccgccc  
accctatTTTTTgagcagatttagtgacctggaaaaaaaaatttgagtacaaaacttttccgaatacaataaaaacggcgggga  
tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccaggggtccga  
gctgagaattggatgcaaaaaaagggttgtccacgcaatcgcgaaaccaacgcggacccaaaggcaagaccgataaaggaga  
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataataaaggaagggcttatag  
gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtggtgggagcgcaa  
cggttttggcccttgtagaggccccgtataaacaaggaggggccaattatgagagagctaatctatcgcggtgcgtgttcat  
aacttgagttaaaaaataggagaccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta  
ttggccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaggagcggaccgaaaggggaag  
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaggagcggga

# Challenge Problem

- Find a motif in a sample of
  - 20 “random” sequences (e.g. 600 nt long)
  - each sequence containing an implanted pattern of length 15,
  - each pattern appearing with 4 mismatches as (15,4)-motif.

# Why Finding (15,4) Motif is Difficult?



# Combinatorial Gene Regulation

- A microarray experiment showed that when gene X is knocked out, 20 other genes are not expressed
  - **How can one gene have such drastic effects?**

# The Motif Finding Problem

- Given a random sample of DNA sequences:

```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcaatctatgcgtttccaacat  
agtactggtgtacatcttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc  
aacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatctt  
agcctccgatgtaagtcatactgtaactattacctgccaccctattacatcttacgtacgtataca  
ctgttatacaacgcgtcatggcgggggtatgcgttttggtcgctcgtacgctcgatcgttaacgtacgtc
```

- Find the pattern that is implanted in each of the individual sequences, namely, the motif

# The Motif Finding Problem (cont'd)

- Additional information:
  - The hidden sequence is of length 8
  - The pattern is not exactly the same in each array because random point mutations may occur in the sequences

# The Motif Finding Problem (cont'd)

- The patterns revealed with no mutations:

cctgatagacgctatctggctatc**acgtacgt**aggtcctctgtgCGaatctatgcggtttccaacat  
agtactgggtgtacatttgat**acgtacgt**acaccggcaacctgaaacaaacgctcagaaccagaagtgc  
aa**acgtacgt**gcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt  
agcctccgatgtaagtcatagctgtaactattacctgccaccctattacatctt**acgtacgt**tataca  
ctgttatacaacgcgtcatggcggggatgcgttttggtcgctcgctacgctcgatcgtt**acgtacgt**c

**acgtacgt**

Consensus String



# The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgCGAATctatgCGTTTTccaacat  
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc  
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaat  
agcctccgatgtaagtcatagctgtaactattacctgccaccctattacatcttacgtCcAtataca  
ctgttatacaacgcgctcatggcggggatgCGTTTTGGTCGTCGTACGCTCGATCGTTACcgtacgGc

# The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgCGaatctatgCGtttccaacat  
agtactgggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc  
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaat  
agcctccgatgtaagtcatagctgtaactattacctgccaccctattacatcttacgtCcAtataca  
ctgttatacaacgcgtcatggcggggatgCGttttggtcgTCgtacgctCGatCGttaCcgtacgGc

Can we still find the motif, now that we have 2 mutations?

# Defining Motifs

- To define a motif, let's say we know where the motif starts in the sequence
- The motif start positions in their sequences can be represented as  $\mathbf{s} = (s_1, s_2, s_3, \dots, s_t)$



# Motifs: Profiles and Consensus

Alignment

a	G	g	t	a	c	T	t
C	c	A	t	a	c	g	t
a	c	g	t	T	A	g	t
a	c	g	t	C	c	A	t
C	c	g	t	a	c	g	G

Profile

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

Consensus

A C G T A C G T

- Line up the patterns by their start indexes

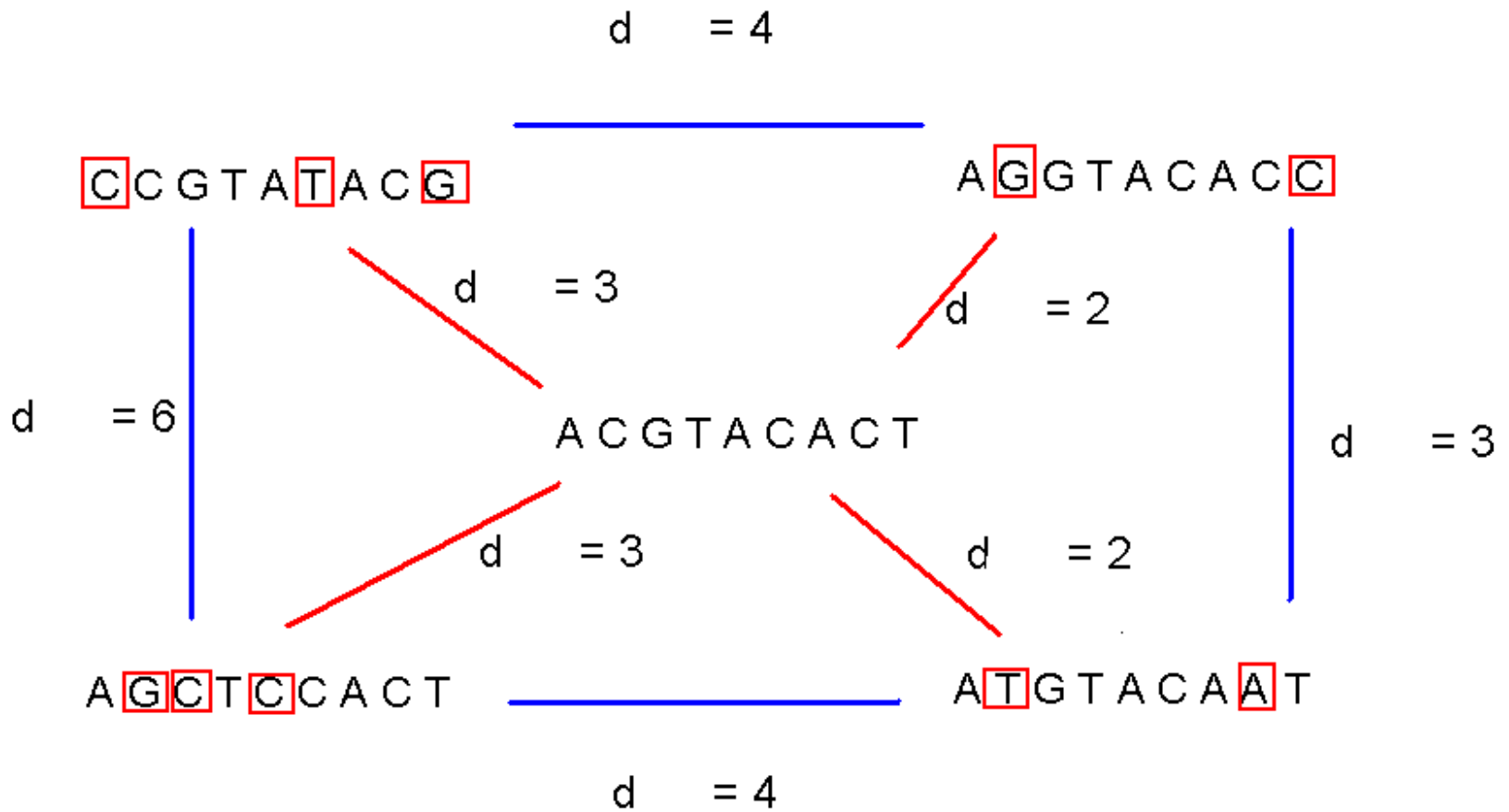
$$\mathbf{s} = (s_1, s_2, \dots, s_t)$$

- Construct matrix profile with frequencies of each nucleotide in columns
- Consensus nucleotide in each position has the highest score in column

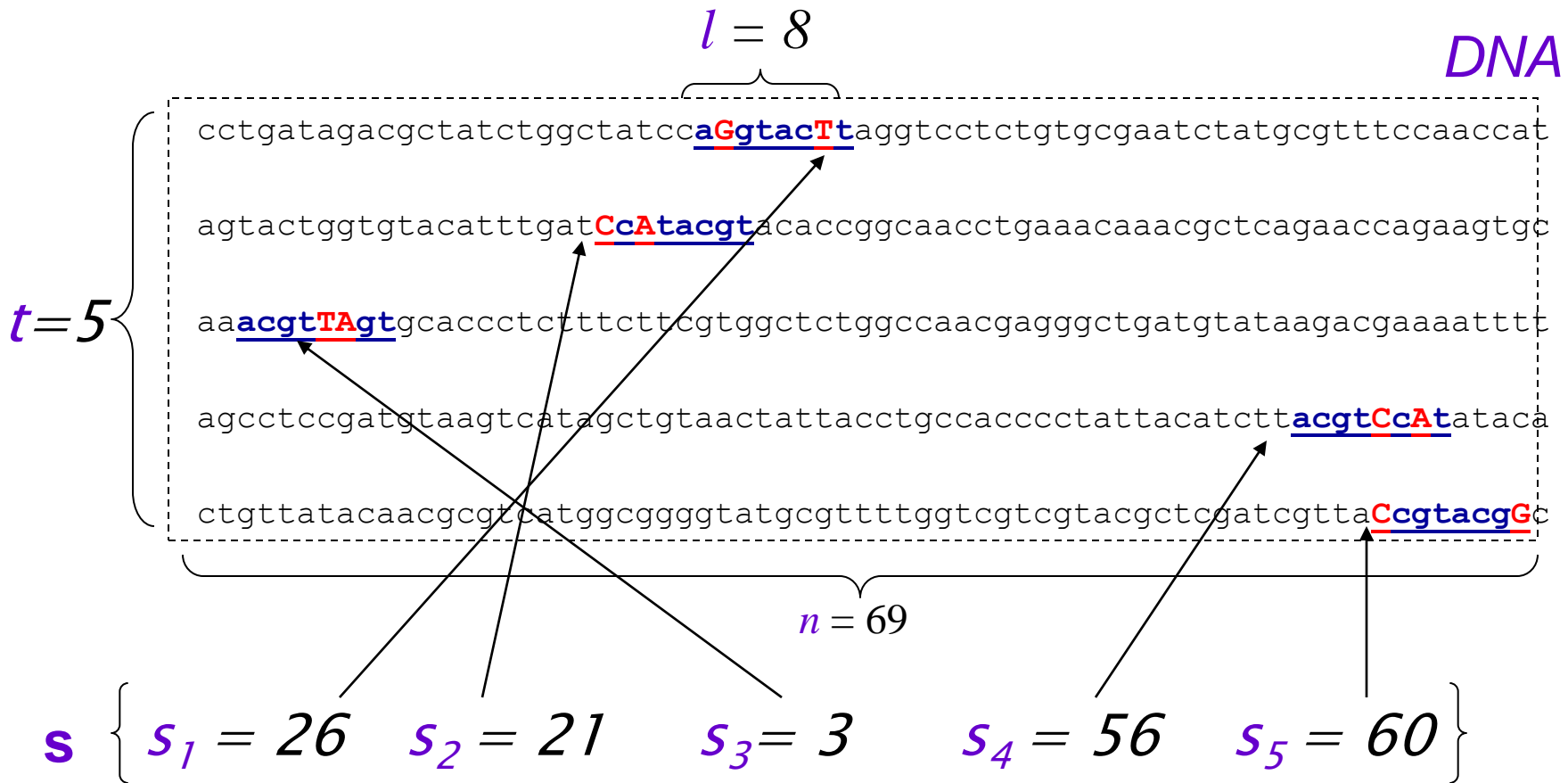
# Consensus

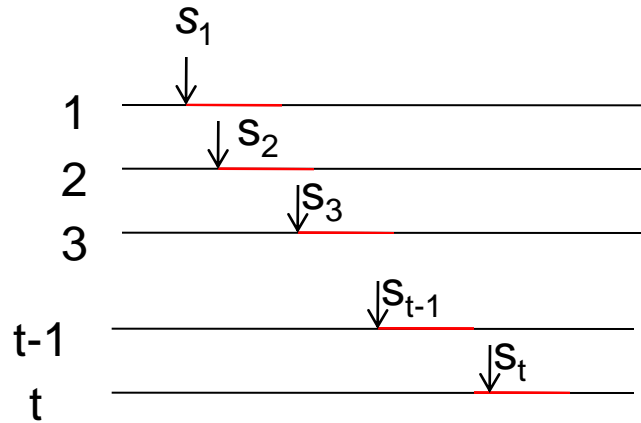
- Think of consensus as an “ancestor” motif, from which mutated motifs emerged
- The *distance* between a real motif and the consensus sequence is generally less than that for two real motifs

# Consensus (cont'd)



# Parameters





Given a group of  $t$  DNA sequences, each of length  $n$ . Find most frequently occurring  $l$ -mers (which are close but differ slightly due to mutations.)

Let  $s=(s_1, s_2, s_3, \dots, s_i, \dots, s_t)$  be the start positions in  $t$  sequences,  $1 \leq s_i \leq n-l+1$ . There are  $(n-l+1)^t$  such starting positions.

```

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

```

Alignment Matrix

```

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

```

Profile matrix  $P(s)$

```

A  T  G  C  A  A  C  T

```

Consensus Sequence  
("most popular")

Not conserved      Highly conserved

Definition:

The score of the alignment for a specific set of positions  $s$  in the  $t$  DNA sequences, denoted as  $Score(s, DNA)$ , is the sum of maximum numbers corresponding to the consensus sequence. For this example this sum is  $5+5+6+4+5+5+6+6=42$



# The Motif Finding Problem: Brute Force Solution I (data driven approach)

The maximum possible  $\text{Score}(\mathbf{s}, \mathbf{DNA}) = lt$  if each column has the same nucleotide and the minimum score is  $(lt/4)$  when each column has  $t/4$  *A, C, G and T*.

- Compute the scores for each possible combination of starting positions  $\mathbf{s}$
- The best score will determine the best profile and the consensus pattern in  $\mathbf{DNA}$
- The goal is to maximize  $\text{Score}(\mathbf{s}, \mathbf{DNA})$  by varying the starting positions  $s_i$

# Evaluating Motifs

- We have a guess about the consensus sequence, but how “good” is this consensus?
- Need to introduce a scoring function to compare different guesses and choose the “best” one.

# Defining Some Terms

- $t$  - number of sample DNA sequences
- $n$  - length of each DNA sequence
- **DNA** - sample of DNA sequences ( $t \times n$  array)
- $l$  - length of the motif ( $l$ -mer)
- $s_i$  - starting position of an  $l$ -mer in sequence  $i$
- $\mathbf{s} = (s_1, s_2, \dots, s_t)$  - array of motif's starting positions

# Scoring Motifs

- Given  $\mathbf{s} = (s_1, \dots, s_t)$  and **DNA**:

$$\text{Score}(\mathbf{s}, \text{DNA}) = \sum_{i=1}^l \max_{k \in \{A, T, C, G\}} \text{count}(k, i)$$

$l$							
<span style="font-size: 2em;">}</span>							
a	G	g	t	a	c	T	t
C	c	A	t	a	c	g	t
a	c	g	t	T	A	g	t
a	c	g	t	C	c	A	t
C	c	g	t	a	c	g	G

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

Consensus a c g t a c g t

Score 3+4+4+5+3+4+3+4=30

# The Motif Finding Problem

- If starting positions  $\mathbf{s}=(s_1, s_2, \dots, s_t)$  are given, finding consensus is easy even with mutations in the sequences because we can simply construct the profile to find the motif (consensus)
- But... the starting positions are usually not given. How can we find the “best” profile matrix?

# The Motif Finding Problem: Formulation

- Goal: Given a set of DNA sequences, find a set of  $l$ -mers, one from each sequence, that maximizes the consensus score
- Input: A  $t \times n$  matrix of **DNA**, and  $l$  the length of the pattern to find
- Output: An array of  $t$  starting positions  $\mathbf{s} = (s_1, s_2, \dots, s_t)$  maximizing **Score( $\mathbf{s}, \text{DNA}$ )**

# BruteForceMotifSearch

1. BruteForceMotifSearch(*DNA*, *t*, *n*, *l*)
2. *bestScore*  $\leftarrow$  0
3. for each  $\mathbf{s}=(s_1, s_2, \dots, s_t)$  from  $(1, 1 \dots 1)$   
to  $(n-l+1, \dots, n-l+1)$
4.     if (*Score*( $\mathbf{s}$ , *DNA*) > *bestScore*)
5.         *bestScore*  $\leftarrow$  *score*( $\mathbf{s}$ , *DNA*)
6.         *bestMotif*  $\leftarrow$   $(s_1, s_2, \dots, s_t)$
7. return *bestMotif*

Line 3 above enumerates all possible  $(n-l+1)^t$  *t*-tuples of position indices in *t* DNA sequences. A systematic method to generate these position indices will be discussed later.

# Running Time of BruteForceMotifSearch

- Varying  $(n - \ell + 1)$  positions in each of  $t$  sequences, we're looking at  $(n - \ell + 1)^t$  sets of starting positions
- For each set of starting positions, the scoring function makes  $\ell$  operations, so complexity is  $\ell(n - \ell + 1)^t = O(\ell n^t)$
- That means that for  $t = 8$ ,  $n = 1000$ ,  $\ell = 10$  we must perform approximately  $10^{20}$  computations – it will take billions years



# The Median String Problem

Another alternative view onto this problem is to reframe the motif finding problem as the Problem of finding a *median string*.

- Given a set of  $t$  DNA sequences find a pattern that appears in all  $t$  sequences with the minimum number of mutations
- This pattern will be the motif

# Hamming Distance

- Hamming distance:
  - $d_H(\mathbf{v}, \mathbf{w})$  is the number of nucleotide pairs that do not match when  $\mathbf{v}$  and  $\mathbf{w}$  are aligned. For example:

$$d_H(\text{AAAAAA}, \text{ACAAAC}) = 2$$

# Total Distance: Example

- Given  $v = \text{"acgtacgt"}$  and  $s$

$$d_H(v, x) = 1 \rightarrow \text{acgtac} \mathbf{g} \mathbf{t}$$

cctgatagacgctatctggctatccacgtac**A**taggtcctctgtgccaatctatgcgtttccaacat

$$d_H(v, x) = 0 \rightarrow \text{acgtacgt}$$

agtactgggtgtacatttgat**acgtacgt**acaccggcaacctgaaacaaacgctcagaaccagaagtgc

**acgtacgt**

aaa**A**gt**C**cgtagcaccctctttcttctgctggctctggccaacgagggctgatgtataagacgaaaat

$$d_H(v, x) = 2$$

agcctccgatgtaagtcataagctgtaactattacctgccaccctattacatctt**acgtacgt**ataca

$$d_H(v, x) = 0 \rightarrow \text{acgtacgt}$$

$$d_H(v, x) = 1 \rightarrow \text{acgtac} \mathbf{g} \mathbf{t}$$

ctggtataacaacgcgctcatggcggggatgcgttttggtcgctcgtagcgtcgatcgтта**acgtac****G**gtc

$v$  is the sequence in red,  $x$  is the sequence in blue

- $TotalDistance(v, DNA) = 1 + 0 + 2 + 0 + 1 = 4$

# Total Distance: Definition

- Given a  $\ell$ -mer  $\mathbf{v}$ , for each DNA sequence  $i$ , compute all  $d_H(\mathbf{v}, \mathbf{x})$ , where  $\mathbf{x}$  is an  $\ell$ -mer with starting position  $s_i$   
( $1 \leq s_i \leq n - \ell + 1$ )
  - Find minimum of  $d_H(\mathbf{v}, \mathbf{x})$  among all  $\ell$ -mers in sequence  $i$
  - *TotalDistance* ( $\mathbf{v}$ , **DNA**) is the sum of the minimum Hamming distances for each DNA sequence  $i$
  - *TotalDistance* ( $\mathbf{v}$ , **DNA**) =  $\min_{\mathbf{s}} d_H(\mathbf{v}, \mathbf{s})$ , where  $\mathbf{s}$  is the set of starting positions  $s_1, s_2, \dots, s_t$
- $\min_{\mathbf{s}} d_H(\mathbf{v}, \mathbf{s}) = \min$  sum of Hamming distances for all choices of  $\ell$ -mers (there are  $4^\ell$  such  $\ell$ -mers ( min of all choices of positions  $\mathbf{s} = s_1, s_2, \dots, s_t$ ))

# The Median String Problem: Formulation

- Goal: Given a set of DNA sequences, find a median string
- Input: A  $t \times n$  matrix  $DNA$ , and  $\ell$ , the length of the pattern to find
- Output: A string  $v$  of  $\ell$  nucleotides that **minimizes**  $TotalDistance(v, DNA)$  over all strings of that length

# Median String Search Algorithm

1. BF\_MedianStringSearch (*DNA*, *t*, *n*, *l*)
2. *bestWord*  $\leftarrow$  AAA...A
3. *bestDistance*  $\leftarrow \infty$  ( or a large number)
4. for each *l*-mer **word** from AAA...A to TTT...T if *TotalDistance*(**word**,*DNA*) < *bestDistance*  
*bestDistance*  $\leftarrow$  *TotalDistance*(**word**,*DNA*)
  1. *bestWord*  $\leftarrow$  **word**
  2. return *bestWord*

Line 4 dominates the computational complexity: we need to enumerate  $4^l$  DNA sequences and for each such sequence, compute the Hamming distance  $n-l+1$  times taking  $l$  comparison operation:  $l(n-l+1)4^l$  which again is exponential.

Given the 'word', we can find

The *TotalDistance(word, DNA)* in a single pass over DNA, that is in time  $O(nt)$ , rather than considering all possible combinations of start positions. Thus, BF median search much less time compared to BF takes  $O(4^l nt)$  time motif search. Typical value of  $l=8$  to 15 and length of upstream motif area is about 500 to 1000 nucleotides.

# Motif Finding Problem == Median String Problem

- The *Motif Finding* is a maximization problem while *Median String* is a minimization problem
- However, the *Motif Finding* problem and *Median String* problem are computationally equivalent
- Need to show that minimizing *TotalDistance* is equivalent to maximizing *Score*



# We are looking for the same thing

Alignment

a	G	g	t	a	c	T	t
C	c	A	t	a	c	g	t
a	c	g	t	T	A	g	t
a	c	g	t	C	c	A	t
C	c	g	t	a	c	g	G

Profile

<b>A</b>	3	0	1	0	3	1	1	0
<b>C</b>	2	4	0	0	1	4	0	0
<b>G</b>	0	1	4	0	0	0	3	1
<b>T</b>	0	0	0	5	1	0	1	4

Consensus

a c g t a c g t

**Score**            3+4+4+5+3+4+3+4

**TotalDistance** 2+1+1+0+2+1+2+1

**Sum**                5 5 5 5 5 5 5 5

- At any column  $i$   
 $Score_i + TotalDistance_i = t$
- Because there are  $l$  columns  
 $Score + TotalDistance = l * t$
- Rearranging:  
 $Score = l * t - TotalDistance$
- $l * t$  is constant. The minimization of the right side is equivalent to the maximization of the left side

# Motif Finding Problem vs. Median String Problem

- Why bother reformulating the Motif Finding problem into the Median String problem?
  - The Motif Finding Problem needs to examine all the combinations for  $\mathbf{s}$ . That is  $(n - \ell + 1)^\ell$  combinations!!!
  - The Median String Problem needs to examine all  $4^\ell$  combinations for  $\mathbf{v}$ . This number is relatively smaller

# Motif Finding: Improving the Running Time

## Recall the BruteForceMotifSearch:

1. BruteForceMotifSearch( $DNA, t, n, l$ )
2.  $bestScore \leftarrow 0$
3. **for each  $s=(s_1, s_2, \dots, s_l)$  from  $(1, 1, \dots, 1)$  to  $(n-l+1, \dots, n-l+1)$**
4.     **if ( $Score(s, DNA) > bestScore$ )**
5.          $bestScore \leftarrow Score(s, DNA)$
6.          $bestMotif \leftarrow (s_1, s_2, \dots, s_l)$
7. return **bestMotif**

# Structuring the Search

- How can we perform the line

for each  $\mathbf{s}=(s_1, s_2, \dots, s_t)$  from  $(1, 1 \dots 1)$  to  $(n-\ell+1, \dots, n-\ell+1)$  ?

- We need a method for efficiently structuring and navigating the many possible motifs
- This is not very different than exploring all  $t$ -digit numbers


# Median String: Improving the Running Time

1. MedianStringSearch (*DNA*, *t*, *n*, *l*)
2. *bestWord*  $\leftarrow$  AAA...A
3. *bestDistance*  $\leftarrow \infty$
4. **for each *l*-mer *s* from AAA...A to TTT...T**  
    **if  $TotalDistance(v, DNA) < bestDistance$**
5.     *bestDistance*  $\leftarrow TotalDistance(s, DNA)$
6.     *bestWord*  $\leftarrow v$
7. **return *bestWord***

While computing  $TotalDistance(v, DNA)$ , it implicitly determines the start positions in the linear scan taking  $O(nt)$  time

# Structuring the Search

- For BruteForce method, we need to sear all possible  $(n - \ell + 1)^t$  start locations.
- For the Median String Problem we need to consider all  $4^\ell$  possible  $\ell$ -mers:

  
aa... aa  
aa... ac  
aa... ag  
aa... at  
.  
.  
tt... tt

How to organize this search?

# Alternative Representation of the Search Space

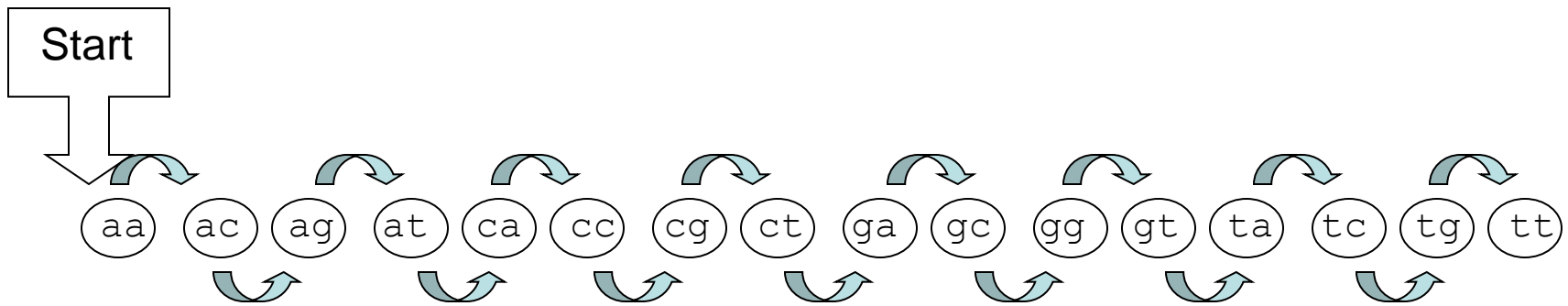
- Let **A** = 1, **C** = 2, **G** = 3, **T** = 4
- Then the sequences from AA...A to TT...T become:

$\ell$   
 $\overbrace{11\dots 11}$   
11...12  
11...13  
11...14  
.  
.  
44...44

- Notice that the sequences above simply list all numbers as if we were counting on base 4 without using 0 as a digit.

# Linked List

Suppose  $l = 2$

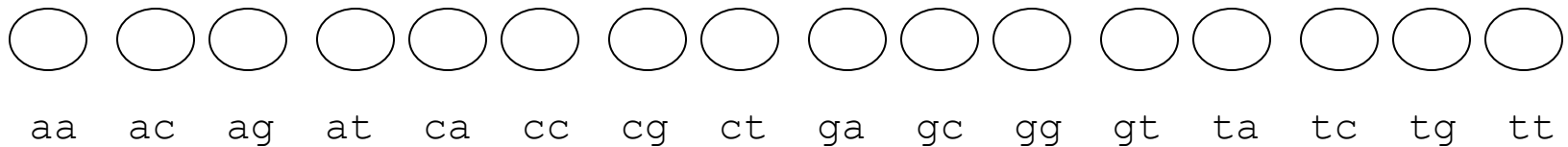


- Need to visit all the predecessors of a sequence before visiting the sequence. It does not allow a subset of the  $l$ -mers to be searched based on some special criteria (viz. Branch and Bound Algorithm). A more efficient data structure is a tree.

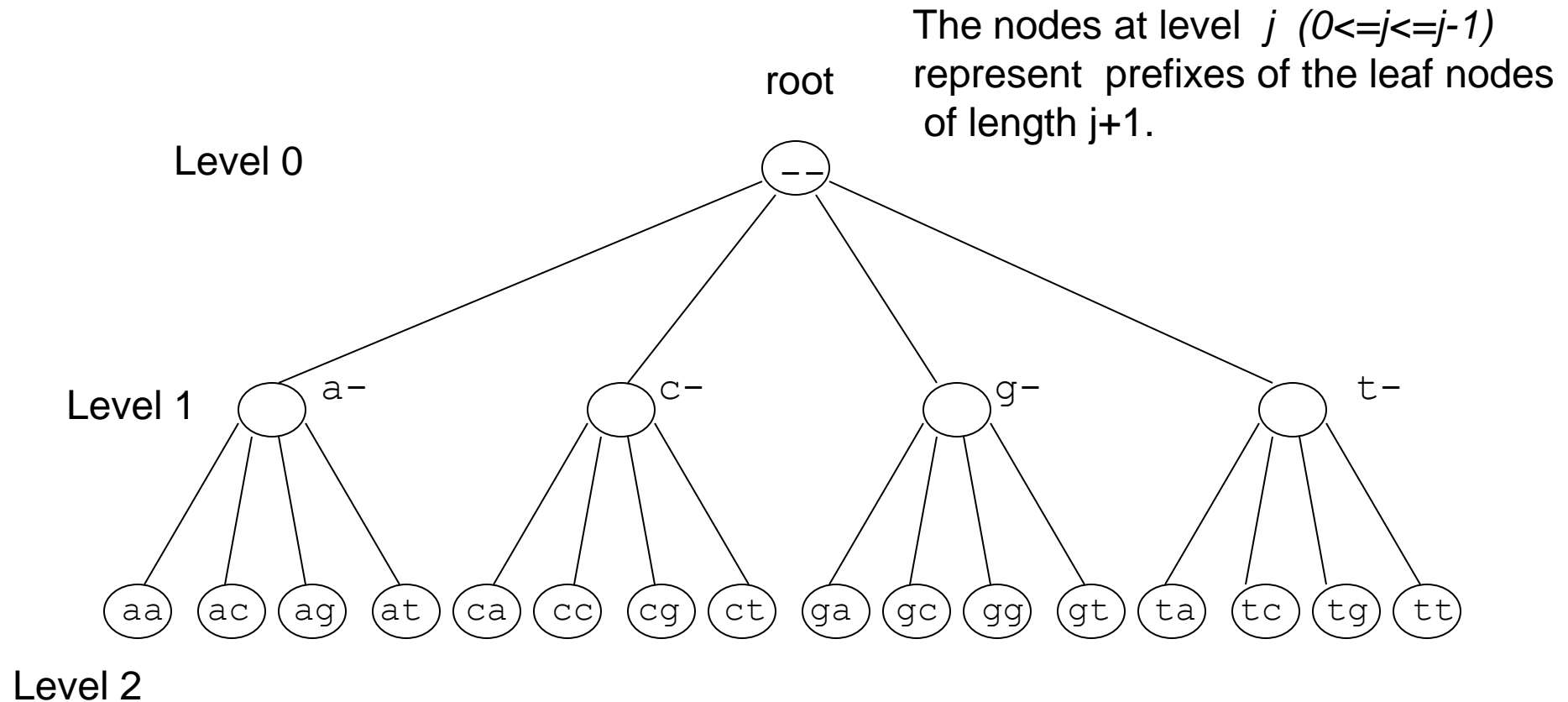


# Linked List (cont'd)

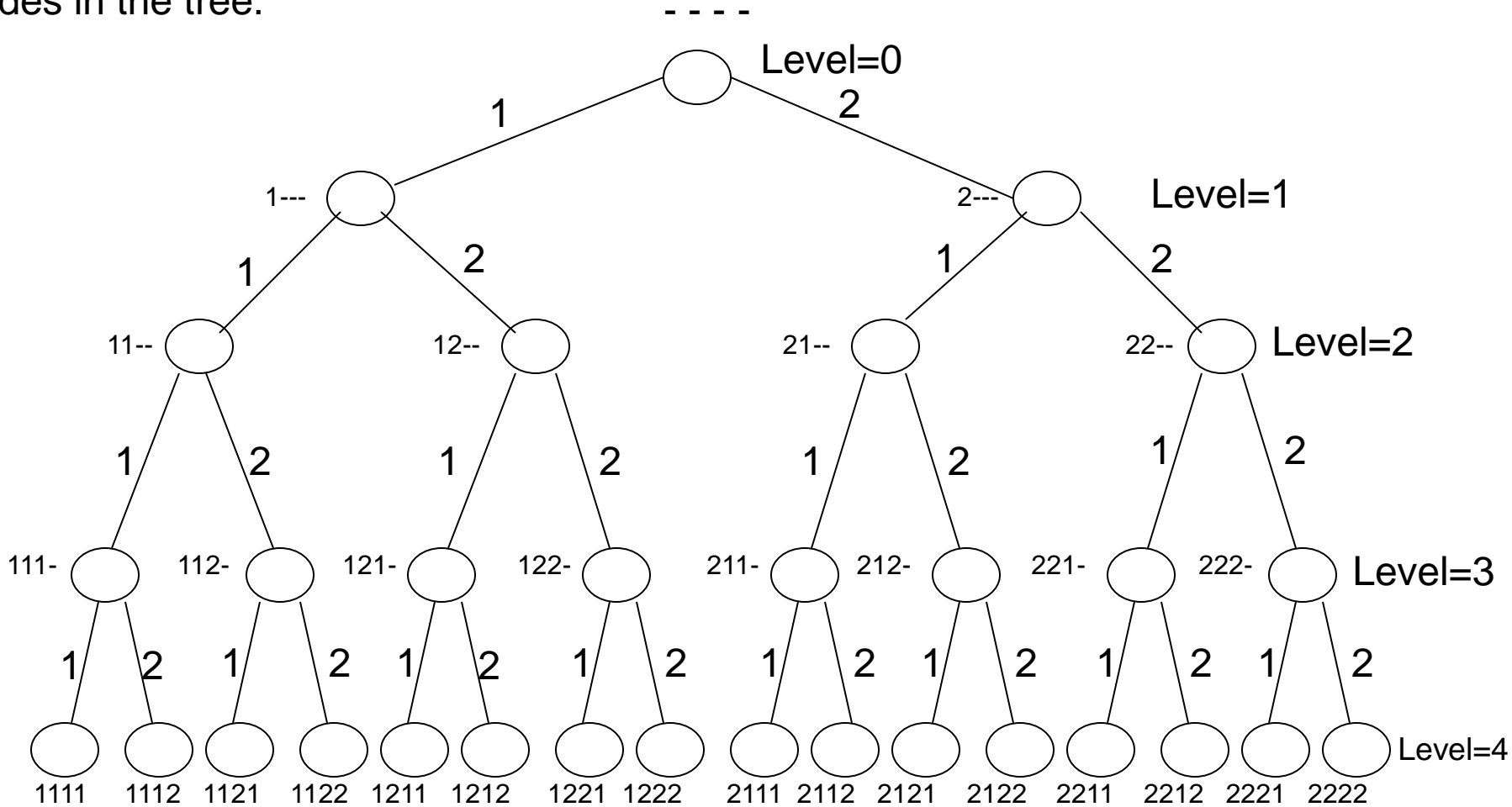
- Linked list is not the most efficient data structure for motif finding
- Let's try grouping the sequences by their prefixes



A Search Tree with and  $k=4$  showing all 2-mers with alphabet (a,c,t,g). The number of levels equals  $\ell+1$



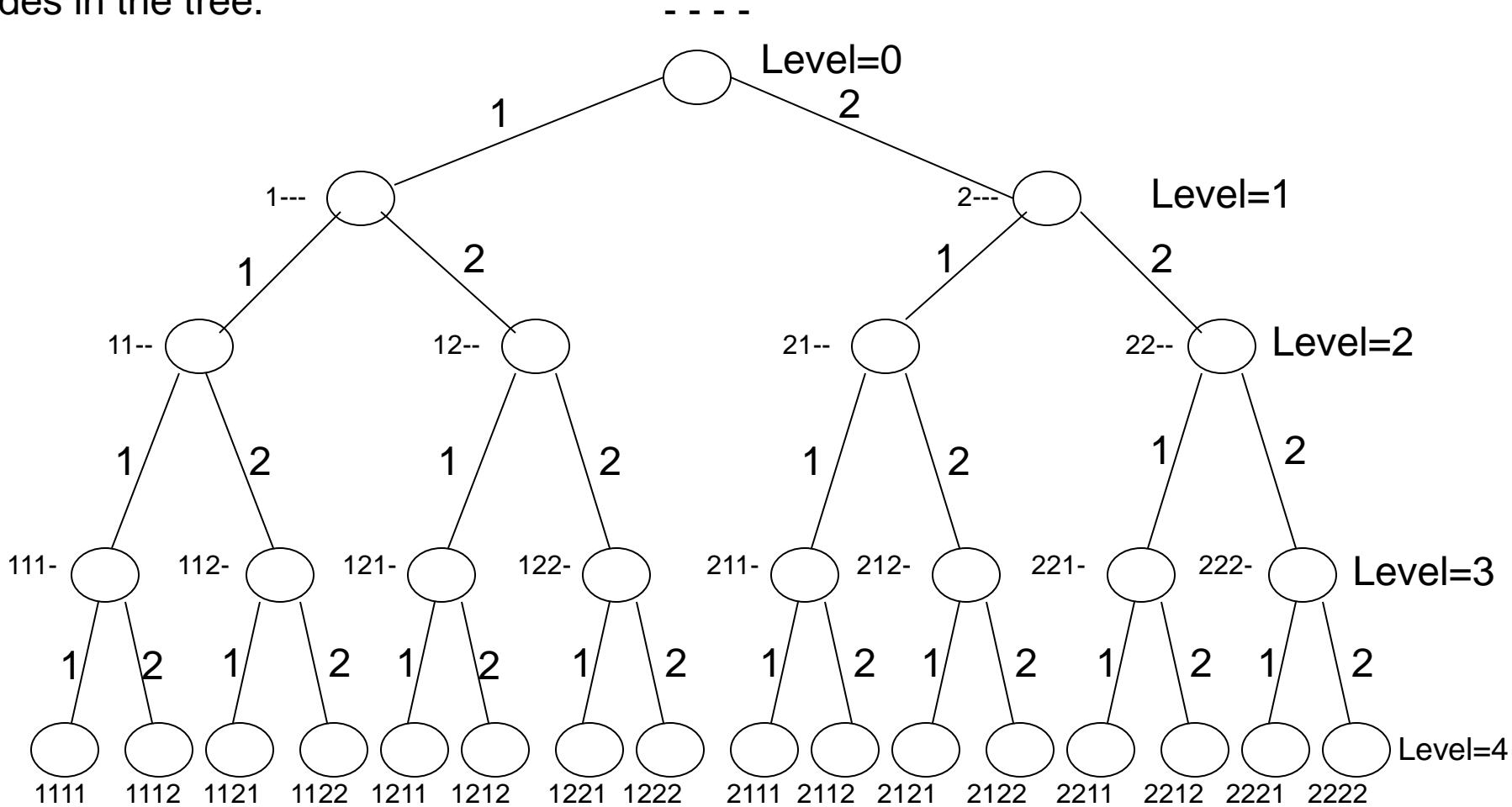
All 4-mers ( $\ell=4$ ) with two letter alphabet (1,2) are represented by the leaves of the tree. An internal node represents all the 4-mers in the leaf nodes of the subtree. The node at level  $v$  ( $0 \leq v \leq 4$ ) is designated by  $v$ -letter prefix of its children. There are  $2^{\ell+1}$  nodes in the tree.



# Analyzing Search Trees

- Characteristics of the search trees:
  - The sequences are contained in its leaves
  - The parent of a node is the prefix of its children
  - To represent all start positions in the Motif Finding problem, we can construct a tree with  $\ell + 1 = t + 1$  levels and each node having  $k = n - \ell + 1$  children for each vertex.
  - For Median String Problem we have all  $\ell$ -mers but  $k = 4$
- How can we move through the tree?

All 4-mers ( $\ell=4$ ) with two letter alphabet (1,2) are represented by the leaves of the tree. An internal node represents all the 4-mers in the leaf nodes of the subtree. A node at level  $v$  ( $0 \leq v \leq 3$ ) is designated by  $v$ -letter prefix of its children. There are  $2^{\ell+1}$  nodes in the tree.



# Moving through the Search Trees

- Four common moves in a search tree that we are about to explore:
  - Move to the next leaf
  - Visit all the leaves
  - Visit the next node
  - Bypass the children of a node
  - In general, we want to consider all  $k^\ell$   $\ell$ -mers. For motif finding problem  $k = n - \ell + 1$ . For median string problem  $k=4$ .

# Moving through the Search Trees

- Four common moves in a search tree that we are about to explore:
  - Move to the next leaf
  - Visit all the leaves
  - Visit the next node
  - Bypass the children of a node
  - In general, we want to consider all  $k^L$  *L*-mers. For motif finding problem  $k = n - \ell + 1$ . For median string problem  $k=4$ .

# Visit the Next Leaf

Given a current leaf  $\mathbf{a}$ , we need to compute the “next” leaf:

NextLeaf ( $\mathbf{a}=(a_1, a_2, \dots, a_L)$ ,  $L$ ,  $k$ )

//  $\mathbf{a}$ : the array of digits.  $L$ : length of the array (it same as  $\ell$ ).  $k$ :  
max

digit value//

1.  $\mathbf{a} \leftarrow \mathbf{a} + 1$  // radix  $k$  addition//

2. if  $\mathbf{a}=(1, 1, \dots, 1)$  exit // no next leaf //  
else return  $\mathbf{a}$

(1, 1, ..., 1, 1)

(4, 4, ..., 3, 3)

(4, 4, ..., 4, 3)

(1, 1, ..., 1, 2)

(4, 4, ..., 3, 3)

(4, 4, ..., 4, 4)

(1, 1, ..., 1, 3)

(4, 4, ..., 3, 4)

(1, 1, ..., 1, 1) // no next leaf//

(1, 1, ..., 1, 4)

(4, 4, ..., 4, 1)

// reset to 1111 //

(1, 1, ..., 2, 1)

.....

.....



# Visit the Next Leaf

Given a current leaf  $\mathbf{a}$ , we need to compute the “next” leaf:

```
1. NextLeaf(  $\mathbf{a}, L, k$  ) //  $\mathbf{a}$  :the array of digits
2. for  $i \leftarrow L$  to 1 //  $L$ : length of the array
3.   if  $a_i < k$  //  $k$ : max digit value
4.      $a_i \leftarrow a_i + 1$ 
5.     return  $\mathbf{a}$ 
6.    $a_i \leftarrow 1$ 
7. return  $\mathbf{a}$ 
```

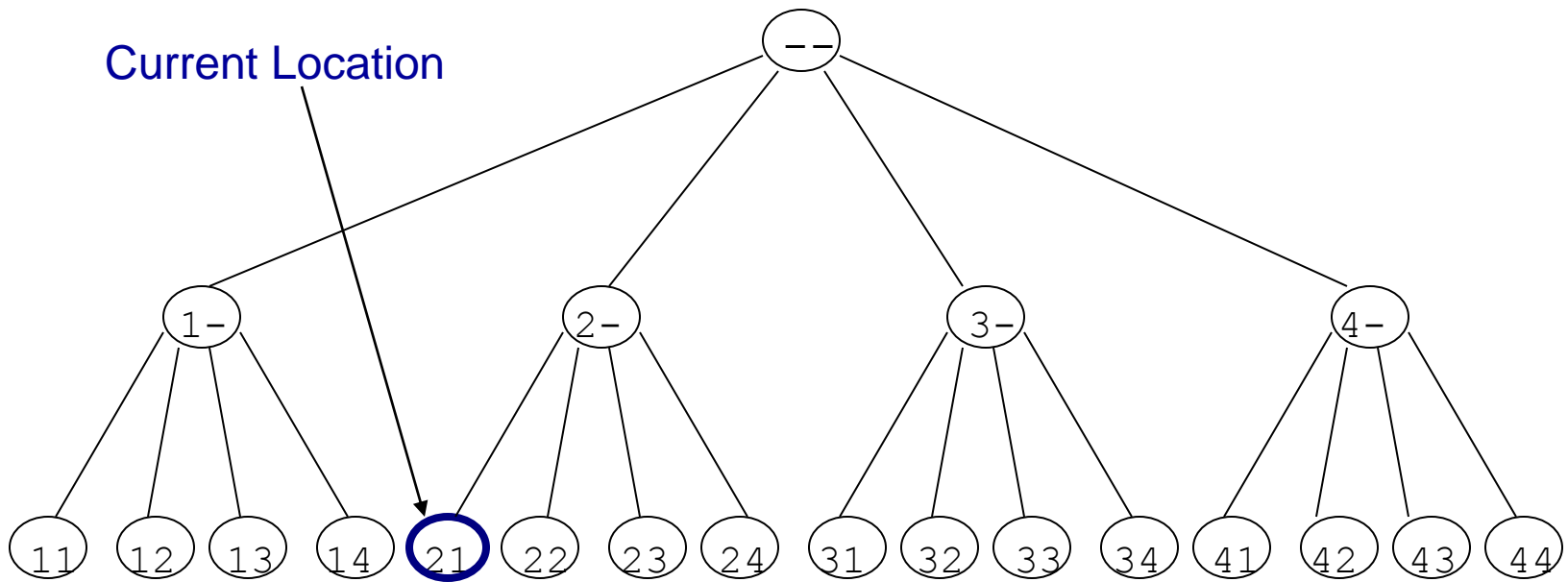
This slide is same as the previous slide but it shows the actual computations performed. If  $L$  were 10, it would be like counting decimal numbers except it uses digits 1 to 10 rather than 0 to 9

# NextLeaf (cont'd)

- The algorithm is addition in radix  $k$
- Increment the least significant digit
- “Carry the one” to the next digit position when the digit is at maximal value

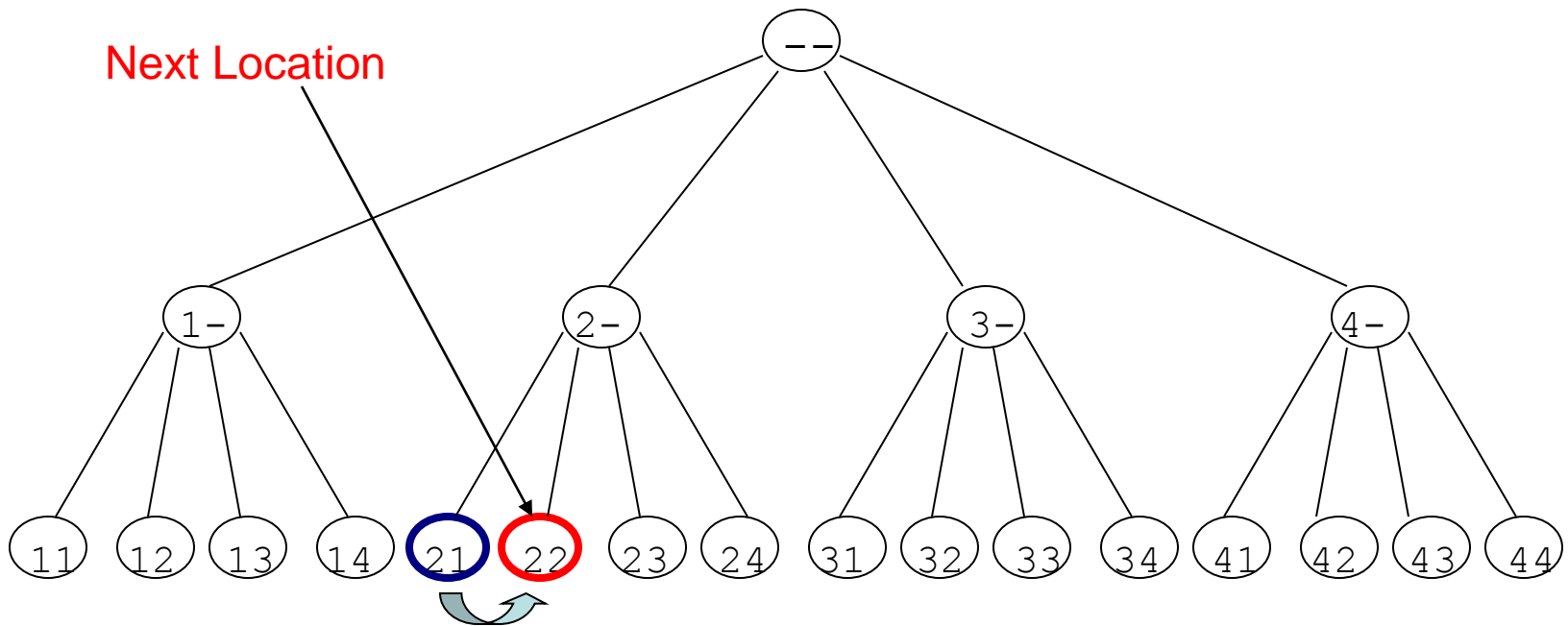
# NextLeaf: Example

- Moving to the next leaf:



# NextLeaf: Example (cont'd)

- Moving to the next leaf:

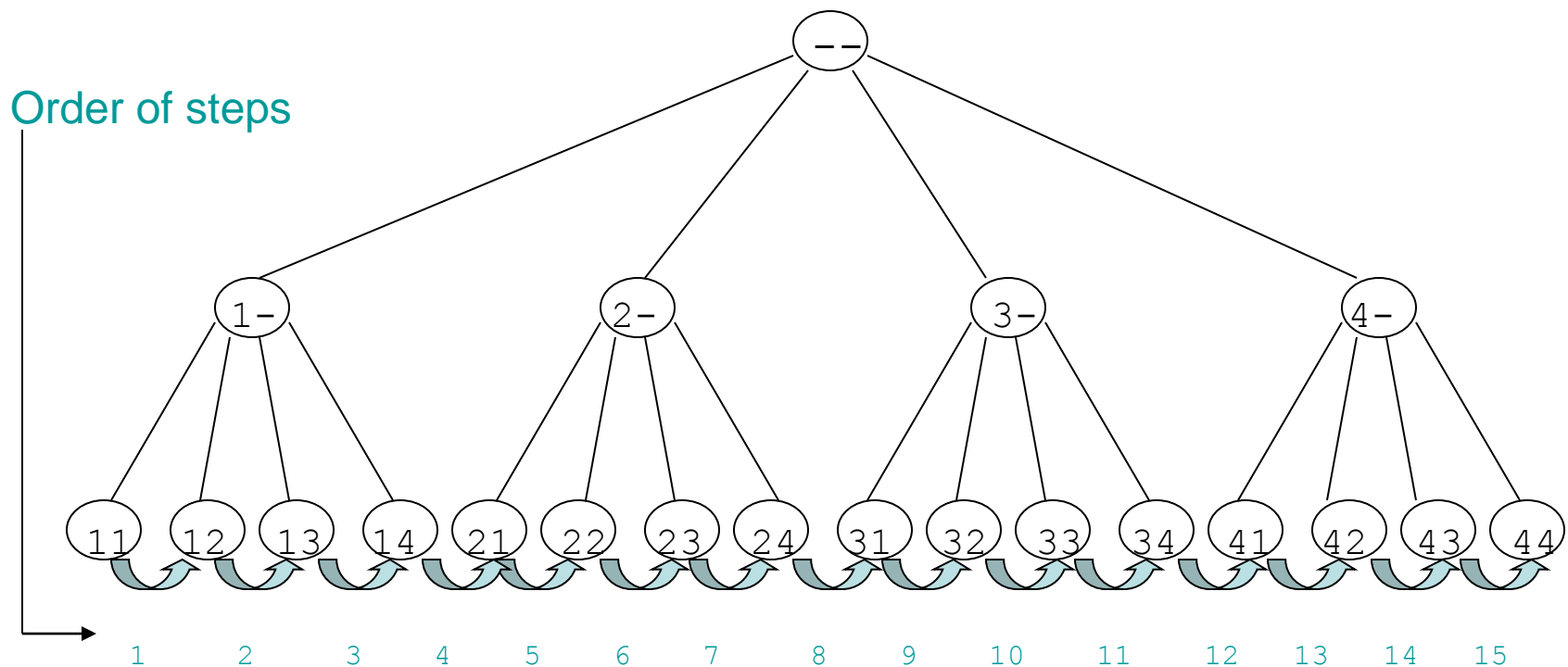


# Visit All Leaves

- Printing all permutations in ascending order:
  1. AllLeaves(L,k) // L: length of the sequence
  2.  $\mathbf{a} \leftarrow (1, \dots, 1)$  // k: max digit value
  3. **while** forever // a : array of digits
  4.     output a
  5.      $\mathbf{a} \leftarrow \text{NextLeaf}(\mathbf{a}, L, k)$
  6.     **if**  $\mathbf{a} = (1, \dots, 1)$
  7.         **return**

# Visit All Leaves: Example

- Moving through all the leaves in order:



# Depth First Search

- So we can search leaves
- How about searching all vertices of the tree?
- We can do this with a *depth first search*.
- Depth First Search (DFS) is a pre-order traversal of a tree. (other traversals are Inorder , Postorder)
- **Preorder (v) // v is a node of the tree//**
- **Output v**
- **If v has children**
- **Preorder (left child of v)**
- **Preorder (right child of v)**

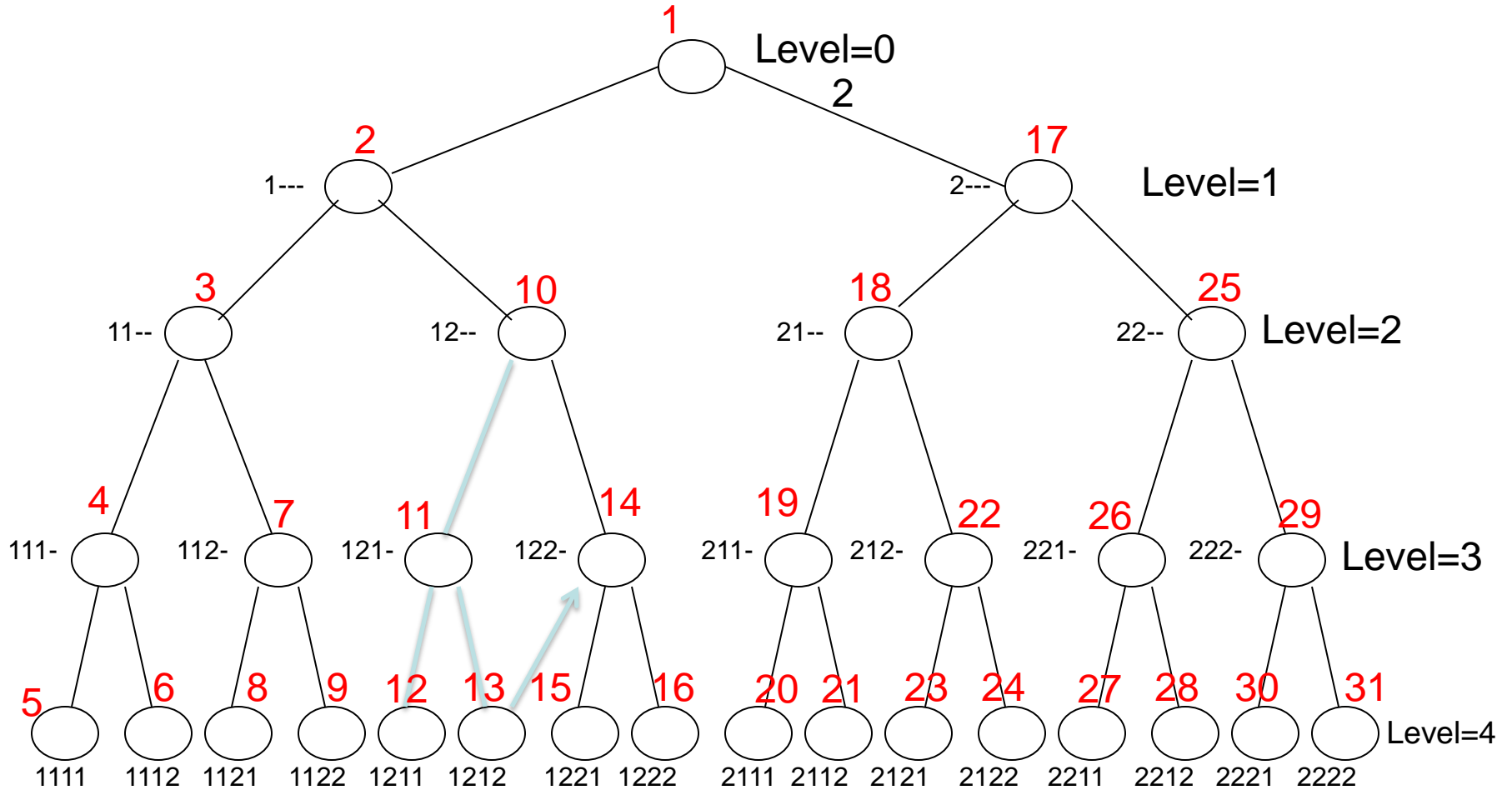
**Preorder (v) // v is a node of the tree//**

**Output v**

**If v has children**

**Preorder (left child of v)**

**Preorder (right child of v)**





# Visit the Next Vertex

## 1. NextVertex( $\mathbf{a}, i, L, k$ )

//  $\mathbf{a} : (a_1, a_2, \dots, a_i \dots a_L)$  – the array of digits. If  $i < L$ , it is an internal vertex.

```
1.  if  $i < L$                                 //  $i$  : prefix length
2.     $a_{i+1} \leftarrow 1$                        //  $L$ : max length
3.    return (  $\mathbf{a}, i+1$  )                   //  $k$ : max digit value
4.  else
5.    for  $j \leftarrow L$  to 1
6.      if  $a_j < k$ 
7.         $a_j \leftarrow a_j + 1$ 
8.        return(  $\mathbf{a}, j$  )
9.  return( $\mathbf{a}, 0$ )
```

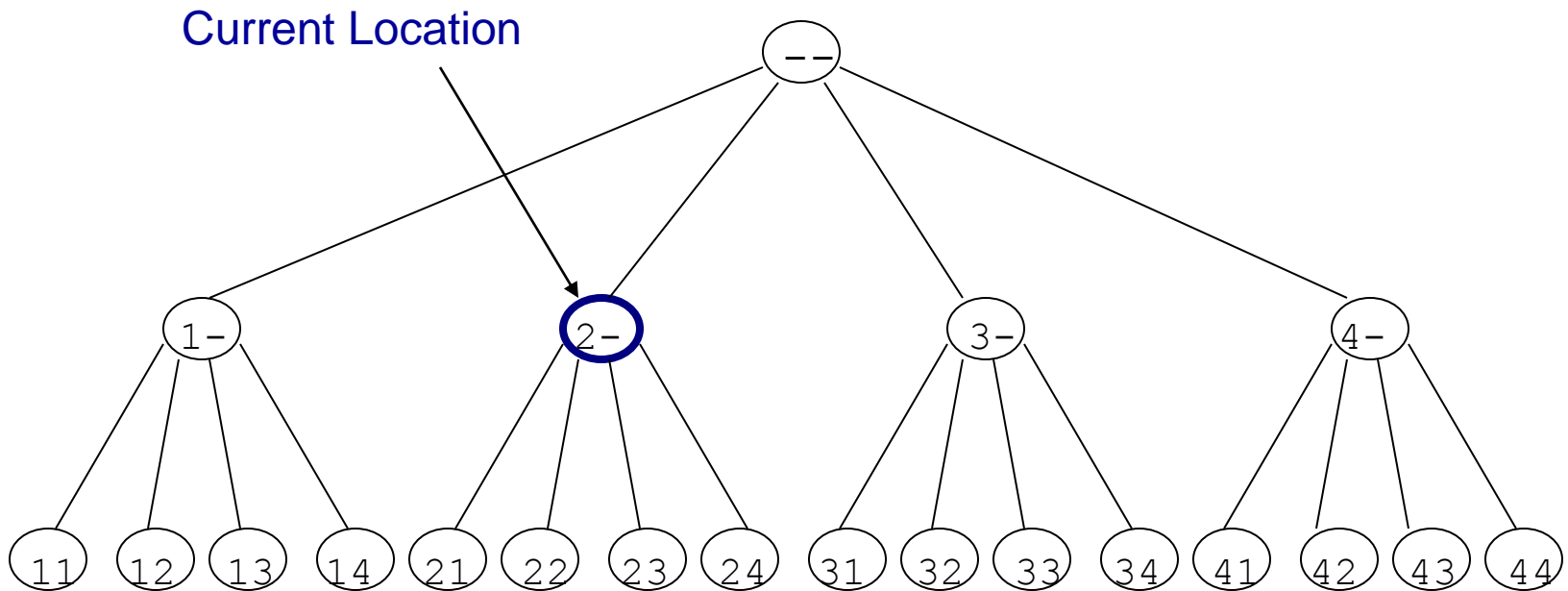
## Explanation

When  $i < L$ , **NextVertex** moves down to next the lower level and explores that subtree of  $a$ . The example in two slides earlier show how the vertices are traversed if the initial vertex specified is 12– (the blue path)

If  $i=L$ , **NextVertex** either moves along the lowest level until the  $L$ -th symbol becomes  $k$  and then jumps back to the right child of the initial vertex if it exists or its right sibling node. It returns to the root  $(a,0)$  after it reaches the vertex  $(k,k,\dots,k)$ .

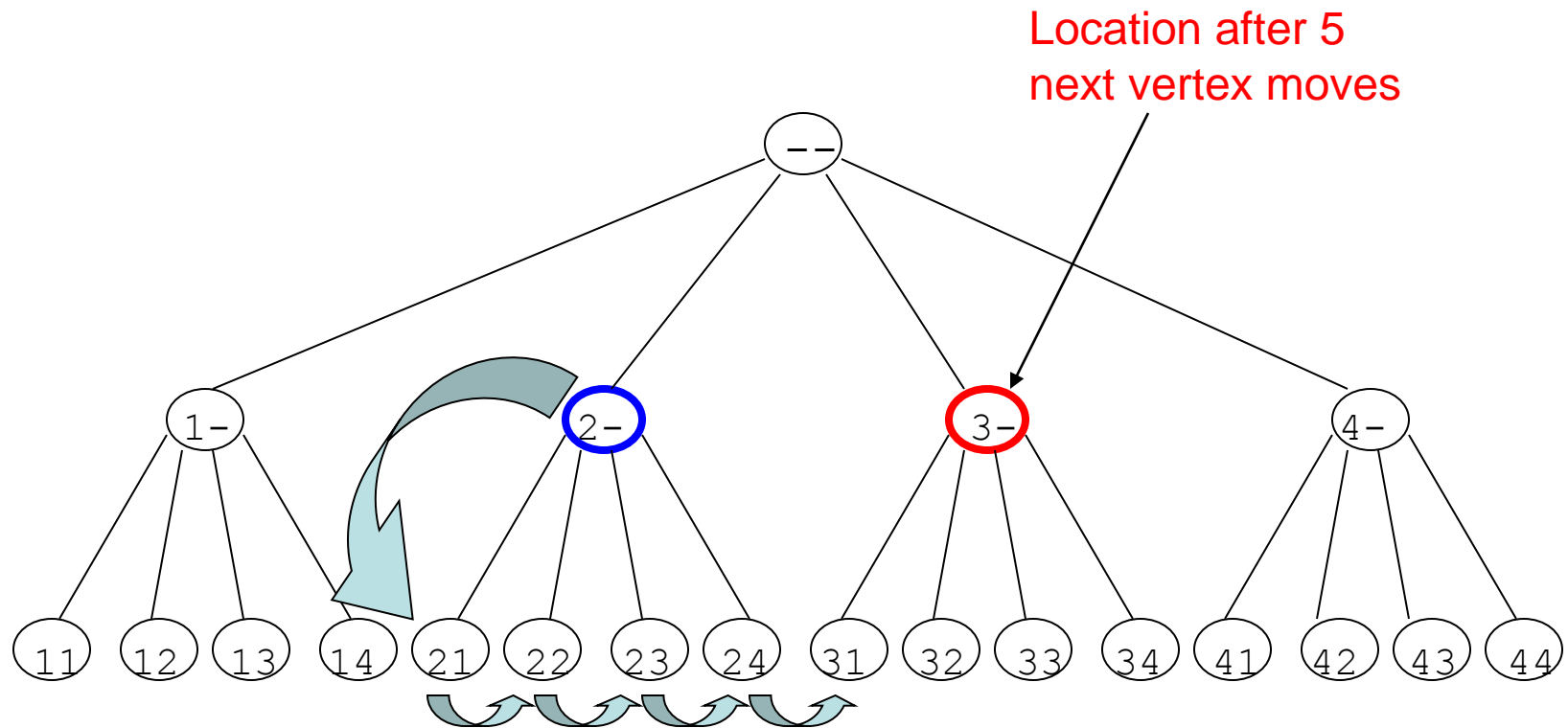
# Example

- Moving to the next vertex:



# Example

- Moving to the next vertices:



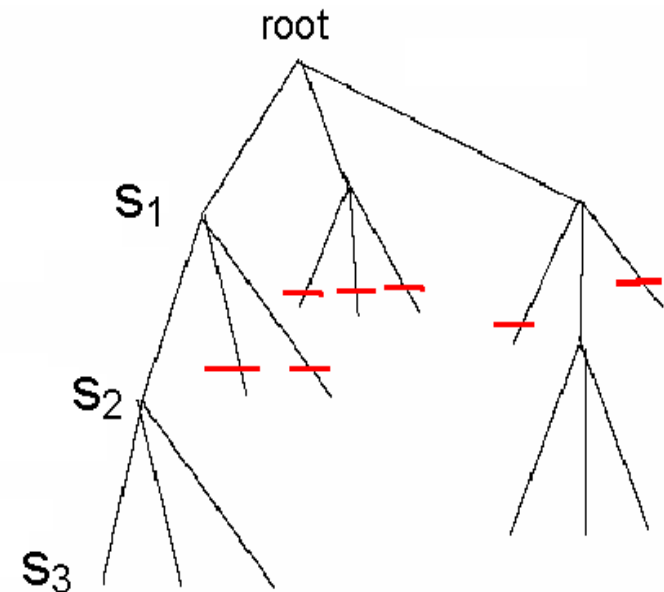
# Branch-and-Bound Search

This approach will allow us to skip any children or grandchildren or great grandchildren etc if these descendants cannot provide a better score than the best leaf that has already been explored.

At each vertex we calculate the most optimistic bound or score of any leaf in the subtree rooted at that vertex and then decide whether to branch further or not. This is the reason why this approach is called **Branch-and-Bound** .

# Branch and Bound Algorithm for Motif Search

- Since each level of the tree goes deeper into search, discarding a prefix discards all following branches. This saves us from looking at  $(n - \ell + 1)^{t-i}$  or  $2^{L-i}$  leaves.
- **NextVertex()** is not up to the task.
  - use **ByPass()** to navigate the tree



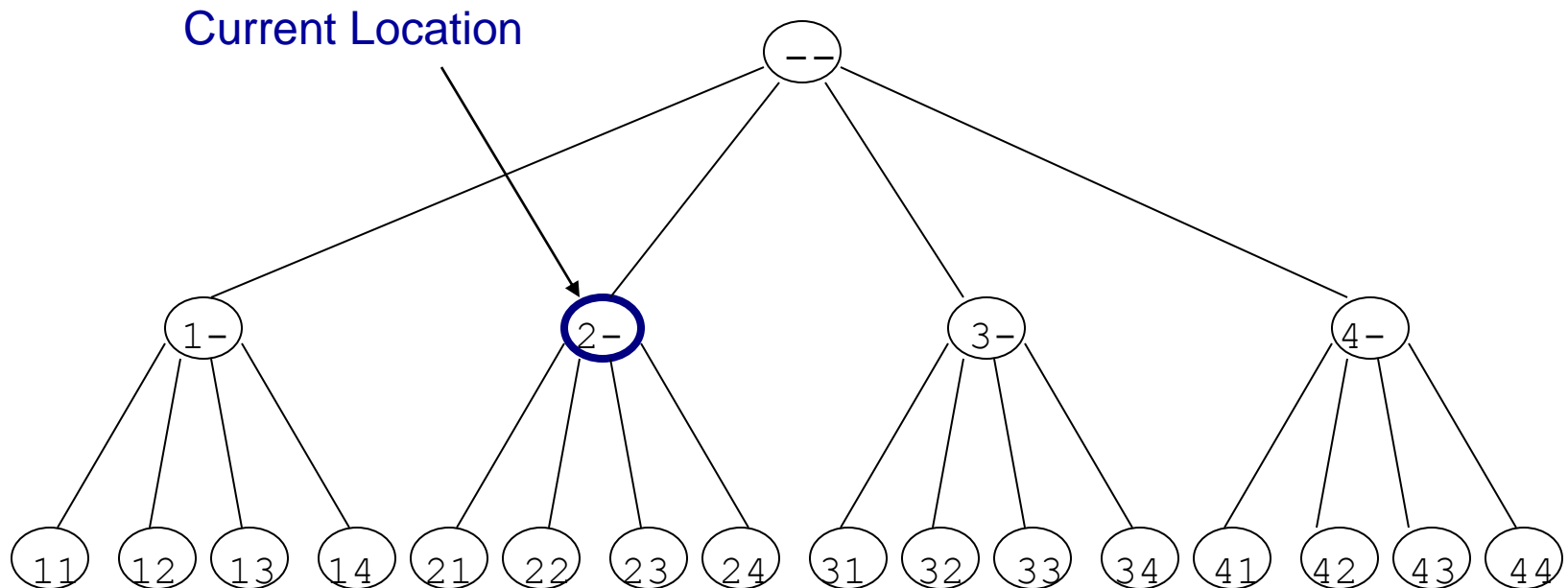
# Bypass Move

- Given a prefix (internal vertex), find next vertex after skipping all its children. If we skip a vertex at level  $i$  of the tree, we can just increment  $a_i$  (unless all vertices at level  $i$  has been traversed and we return to the root vertex). The algorithm looks very similar to NextLeaf but differ in a subtle way.

```
1.  Bypass(a, i, L, k)    // a: array of digits
2.  for  $j \leftarrow i$  to 1  // i: prefix length
3.    if  $a_j < k$            // L: maximum length
4.       $a_j \leftarrow a_j + 1$  // k: max digit value
5.    return(a, j)
6.  return(a, 0)
```

# Bypass Move: Example

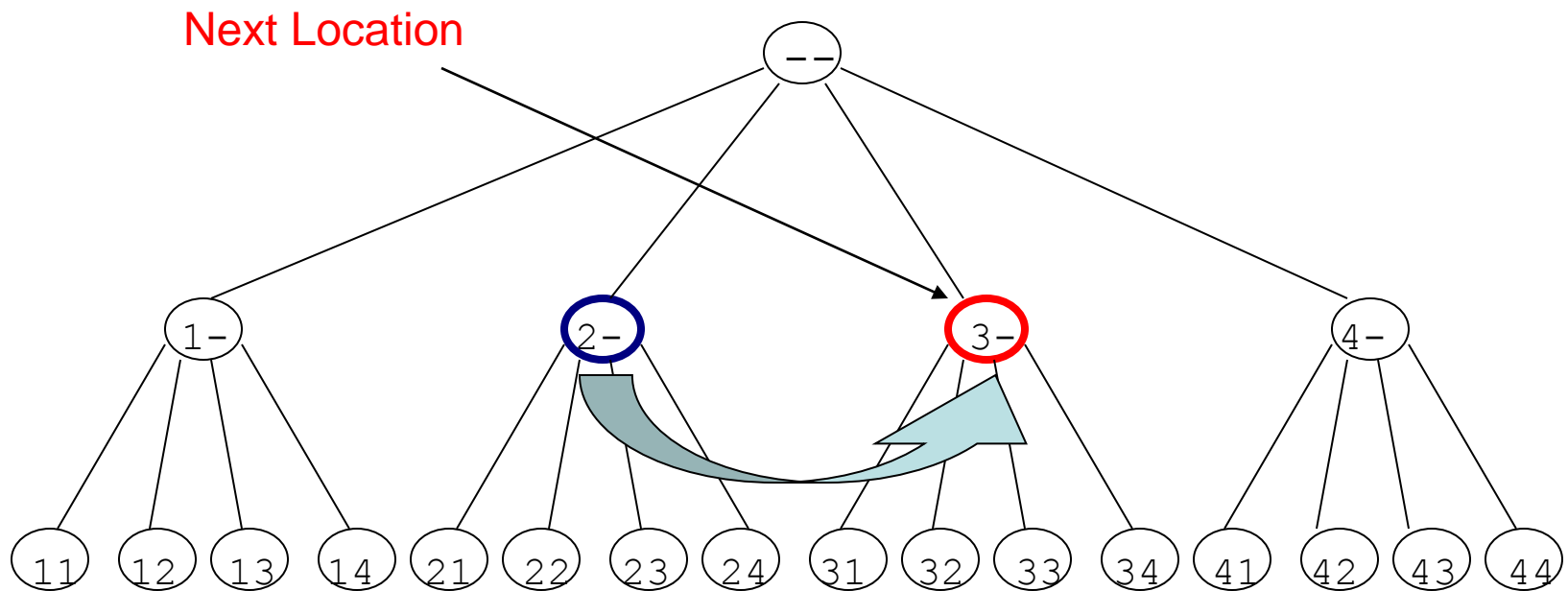
- Bypassing the descendants of “2-”:





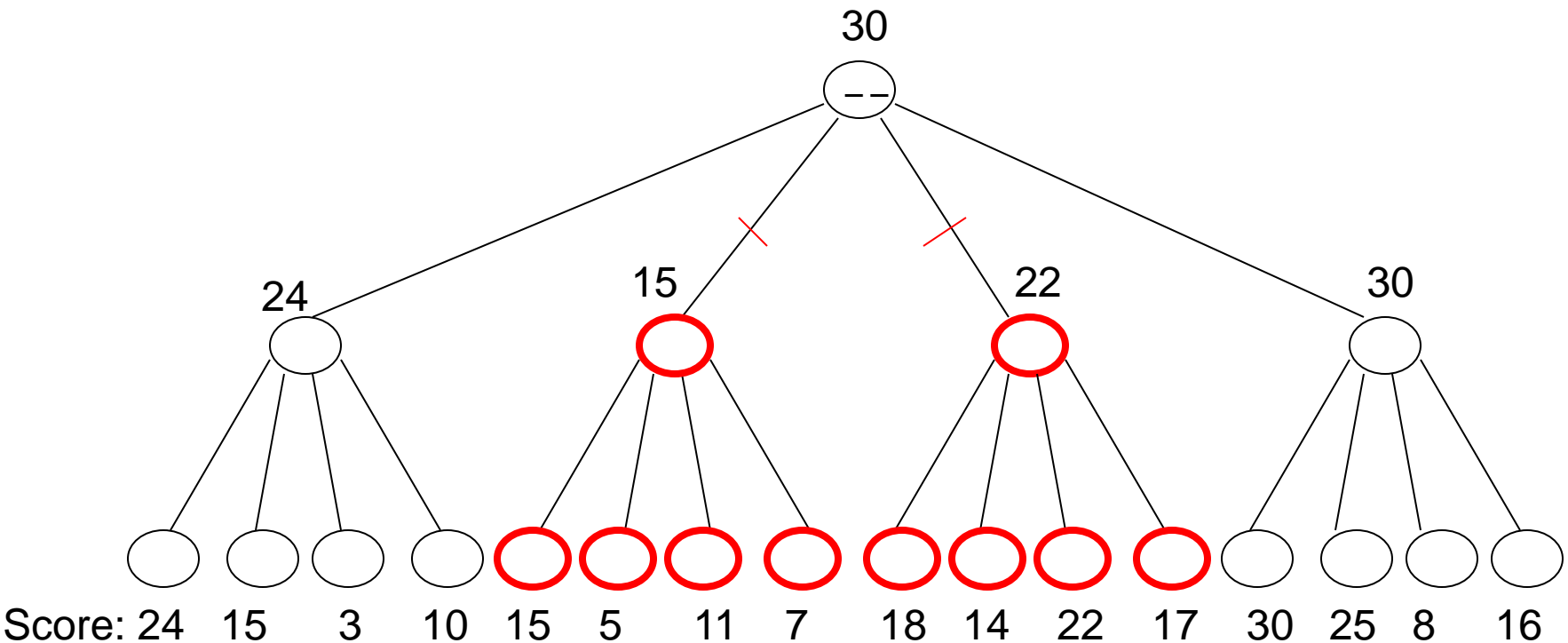
# Example

- Bypassing the descendants of “2-”:



# Revisiting Search Tree

If the scores of the leaf nodes are as given below then it does not make Any sense to explore any subtree whose score is less than 24 (the red nodes)



# Brute Force Search Again

1. BruteForceMotifSearchAgain(*DNA*, *t*, *n*, *l*)
2.  $\mathbf{s} \leftarrow (1, 1, \dots, 1)$
3.  $\mathit{bestScore} \leftarrow \mathit{Score}(\mathbf{s}, \mathit{DNA})$
4. while forever
5.      $\mathbf{s} \leftarrow \mathit{NextLeaf}(\mathbf{s}, t, n - l + 1)$   
[Enumeration of  $(1, 1, \dots, 1)$  to  $(n-l+1, \dots, n-l+1)$ ]
6.     if  $(\mathit{Score}(\mathbf{s}, \mathit{DNA}) > \mathit{bestScore})$
7.          $\mathit{bestScore} \leftarrow \mathit{Score}(\mathbf{s}, \mathit{DNA})$
8.          $\mathit{bestMotif} \leftarrow (s_1, s_2, \dots, s_t)$
9.     if  $\mathbf{s} = (1, 1, \dots, 1)$
10.         return  $\mathit{bestMotif}$

As we discussed earlier, we have to explore  $(n-l+1)^t$  positions. For each position vector  $\mathbf{s} = s_1, s_2, \dots, s_t$ , the algorithm calculates  $\mathit{Score}(\mathbf{s}, \mathit{DNA})$  which takes  $O(l)$  operations. Thus, the overall complexity of the algorithm is  $O(l n^t)$ .

# Can We Do Better?

- A set of start locations  $\mathbf{s}=(s_1, s_2, \dots, s_t)$  may have a weak profile for the first  $i$  positions  $(s_1, s_2, \dots, s_i)$ . Then, it is not necessary to consider ant starting positions beginning  $s_{i-1}, s_{i+1}, \dots, s_t$
- Define **partial consensus score**,  $Score(\mathbf{s}, i, DNA)$  to be the consensus score of the  $i \times l$  alignment matrix involving the first  $i$  rows of  $DNA$  corresponding to starting positions  $(s_1, s_2, \dots, s_i, -, \dots)$ .
- Every row of alignment may add at most  $l$  to **Score**. All subsequent  $(t-i)$  positions  $(s_{i+1}, \dots, s_t)$  could add in the most optimistic situation  $(t-i) * l$  to  $Score(\mathbf{s}, i, DNA)$
- If  $Score(\mathbf{s}, i, DNA) + (t-i) * l < \mathbf{BestScore}$ , it makes no sense to explore the remaining  $(t-i)$  positions in this choice of positions  $(s_1, s_2, \dots, s_i, -, \dots)$ .
- Using **ByPass()** , we could save looking at  $(n-l+1)^{t-i}$  leaves

# Pseudocode for Branch and Bound Motif Search

```
1.  BranchAndBoundMotifSearch(DNA, t, n, l )
2.  s  $\leftarrow$  (1,...,1)
3.  bestScore  $\leftarrow$  0
4.  i  $\leftarrow$  1
5.  while i > 0
6.    if i < t
7.      optimisticScore  $\leftarrow$  Score(s, i, DNA) + (t - i) * l
8.      if optimisticScore < bestScore
9.        (s, i)  $\leftarrow$  Bypass(s, i, t, n-l+1) //parameters (a, i, L, k) //
10.     else
11.       (s, i)  $\leftarrow$  NextVertex(s, i, t, n-l+1) //parameters (a, i, L, k)
12.     else
13.       if Score(s, DNA) > bestScore
14.         bestScore  $\leftarrow$  Score(s, DNA)
15.         bestMotif  $\leftarrow$  (s1, s2, s3, ..., si)
16.         (s, i)  $\leftarrow$  NextVertex(s, i, t, n-l+1) //parameters (a, i, L, k)
17.  return bestMotif
```

# Median String Search Improvements

- Recall the computational differences between motif search and median string search
  - The Motif Finding Problem needs to examine all  $(n-l+1)^t$  combinations for  $s$ .
  - The Median String Problem needs to examine  $4^l$  combinations of  $\nu$ . This number is relatively small
- We want to use median string algorithm with the Branch and Bound trick!

# Branch and Bound Applied to Median String Search

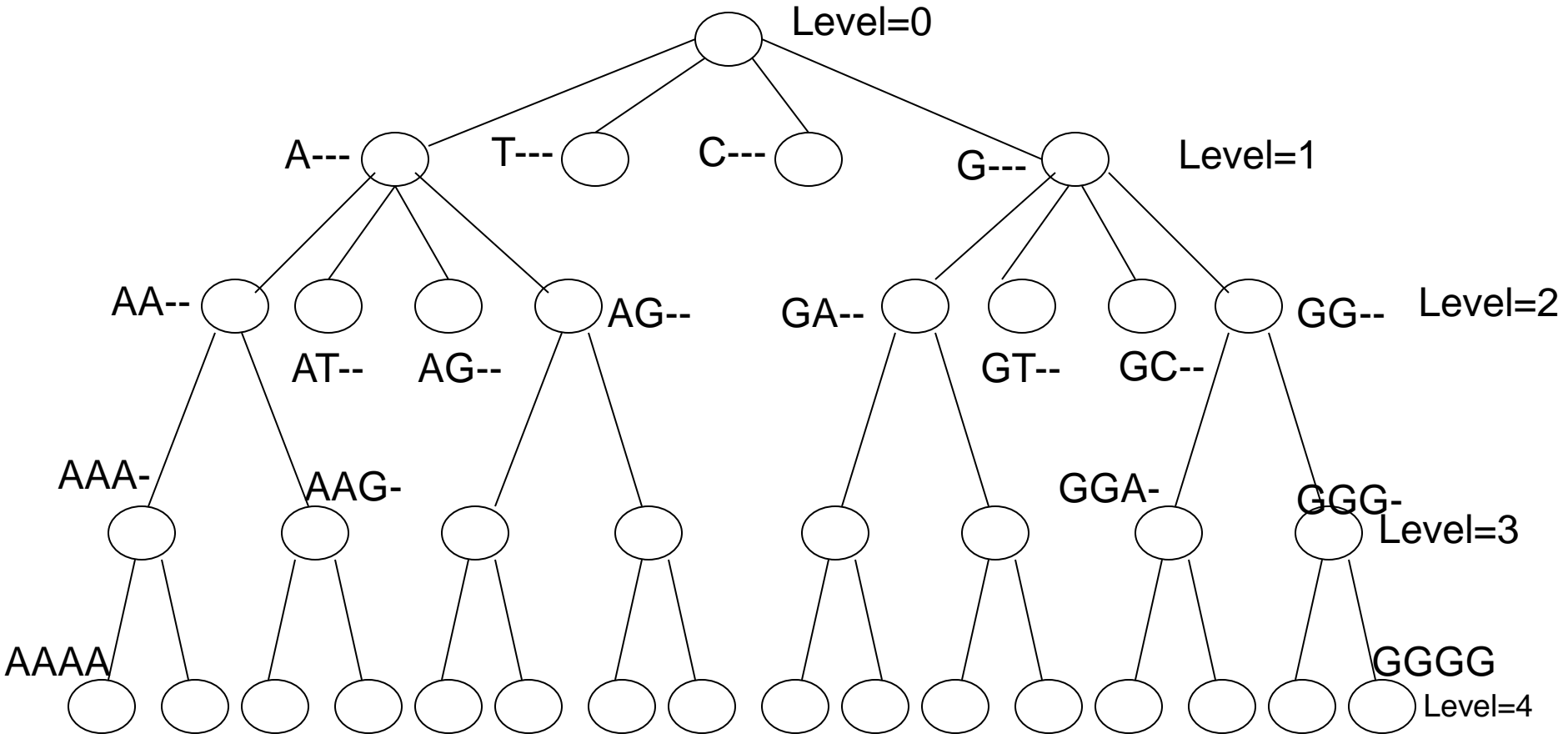
- Note that if the total distance for a prefix is greater than that for the best word so far:

$$\text{TotalDistance}(\textit{prefix}, \textit{DNA}) > \textit{BestDistance}$$

there is no use exploring the remaining part of the word

- We can eliminate that branch and BYPASS exploring that branch further.

A vertex at level  $i$  in this tree represents a nucleotide string of length  $i$  which can be viewed as the  $i$ -long prefix of every leaf below that vertex. The *SIMPLE\_MEDIAN\_SEARCH* pseudocode given in next page maps A,C,G,T to numerals 1,2,3,4 respectively.





# Pseudocode for Branch and Bound Median Search

## SIMPLE\_MEDIAN\_SEARCH(*DNA*, *t*, *n*, *l* )

1.  $\mathbf{s} \leftarrow (1, \dots, 1)$
2.  $\mathbf{bestDistance} \leftarrow$  a very large number
3.  $i \leftarrow 1$
4. **while**  $i > 0$
5.     **if**  $i < \underline{l}$
6.      $(\mathbf{s}, \mathbf{i}) \leftarrow \mathbf{NextVertex}(\mathbf{s}, i, l, 4)$      //parameters (*a*, *i*, *L*, *k*)
7.     **else**
8.          $\mathbf{word} \leftarrow$  nucleotide string corresponding to  $(s_1, s_2, \dots, s_l)$
9.         **if**  $\mathbf{TotalDistance}(\mathbf{word}, \mathbf{DNA}) < \mathbf{bestDistance}$
10.              $\mathbf{bestDistance} \leftarrow \mathbf{TotalDistance}(\mathbf{word}, \mathbf{DNA})$
11.              $\mathbf{bestWord} \leftarrow \mathbf{word}$
12.          $(\mathbf{s}, \mathbf{i}) \leftarrow \mathbf{NextVertex}(\mathbf{s}, i, l, 4)$      //parameters (*a*, *i*, *L*, *k*)
13. **return**  $\mathbf{bestWord}$

We find bound for  $\mathbf{TotalDistance}(\mathbf{word}, \mathbf{DNA})$  at each vertex. If the total distance between the *i*-prefix of the **word** and *DNA* is larger than the smallest seen so far for one of the leaves (nucleotide strings of length *l*), then there is no point investigating subtrees of the vertex corresponding to that *i*-prefix of the **word**; all extensions of this prefix into an *l*-mer will have at least the same or probably more total distance. Of course, there could be some extensions to the prefix that matches every string in the sample which will make the value of total distance to be 0.

# Bounded Median String Search

```
1. BranchAndBoundMedianStringSearch(DNA,t,n,l)
2.  $s \leftarrow (1, \dots, 1)$ 
3.  $bestDistance \leftarrow \infty$ 
4.  $i \leftarrow 1$ 
5. while  $i > 0$ 
6.   if  $i < l$ 
7.      $prefix \leftarrow$  string corresponding to the first  $i$  nucleotides of  $s$ 
8.      $optimisticDistance \leftarrow TotalDistance(prefix, DNA)$ 
9.     if  $optimisticDistance > bestDistance$ 
10.       $(s, i) \leftarrow Bypass(s, i, l, \mathcal{A})$ 
11.     else
12.       $(s, i) \leftarrow NextVertex(s, i, l, \mathcal{A})$ 
13.   else
14.      $word \leftarrow$  nucleotide string corresponding to  $s$ 
15.     if  $TotalDistance(s, DNA) < bestDistance$ 
16.       $bestDistance \leftarrow TotalDistance(word, DNA)$ 
17.       $bestWord \leftarrow word$ 
18.      $(s, i) \leftarrow NextVertex(s, i, l, \mathcal{A})$ 
19. return  $bestWord$ 
```

# Improving the Bounds

- Given an  $\ell$ -mer  $w$ , divided into two parts at point  $i$ 
  - $u$  : prefix  $w_1, \dots, w_i$ ,
  - $v$  : suffix  $w_{i+1}, \dots, w_\ell$
- Find minimum distance for  $u$  in a sequence
- No instances of  $u$  in the sequence have distance less than the minimum distance
- Note this doesn't tell us anything about whether  $u$  is part of any motif. We only get a minimum distance for prefix  $u$

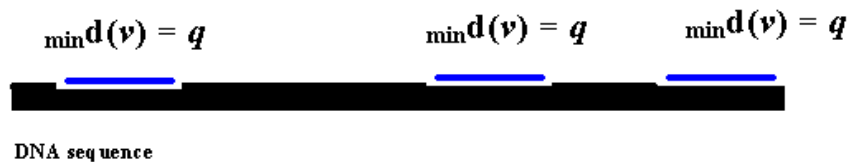
# Improving the Bounds (cont'd)

- Repeating the process for the suffix  $\mathbf{v}$  gives us a minimum distance for  $\mathbf{v}$
- Since  $\mathbf{u}$  and  $\mathbf{v}$  are two substrings of  $\mathbf{w}$ , and included in motif  $\mathbf{w}$ , we can assume that the minimum distance of  $\mathbf{u}$  plus minimum distance of  $\mathbf{v}$  can only be less than the minimum distance for  $\mathbf{w}$

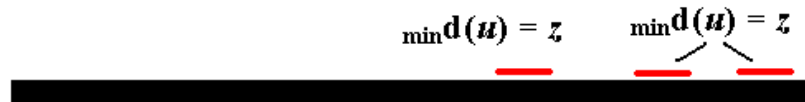
# Better Bounds

Searching for prefix  $V$

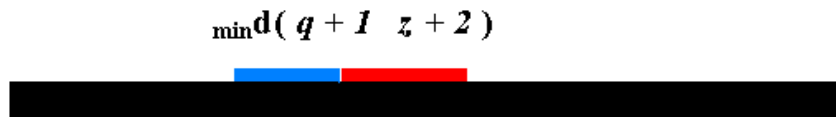
We may find many instances of prefix  $V$  with a minimum distance  $q$



Likewise for  $U$



But for  $U$  and  $V$  combined,  $U$  is not at its minimum distance location, neither is  $V$



But at least we know  $w$  (prefix  $u$  suffix  $v$ ) cannot have distance *less* than  $\min d(v) + \min d(u)$

# Better Bounds (cont'd)

- If  $d(\mathit{prefix}) + d(\mathit{suffix}) \geq \mathit{bestDistance}$ :
  - Motif  $w$  ( $\mathit{prefix.suffix}$ ) cannot give a better (lower) score than  $d(\mathit{prefix}) + d(\mathit{suffix})$
  - In this case, we can **ByPass()**

# Better Bounded Median String Search

```
1. ImprovedBranchAndBoundMedianString(DNA, t, n, l)
2.   s = (1, 1, ..., 1)
3.   bestDistance =  $\infty$ 
4.   i = 1
5.   while i > 0
6.     if i < l
7.       prefix = nucleotide string corresponding to ( $s_1, s_2, s_3, \dots, s_i$ )
8.       optimisticPrefixDistance = TotalDistance(prefix, DNA)
9.       if (optimisticPrefixDistance < bestsubstring[ i ])
10.        bestsubstring[ i ] = optimisticPrefixDistance
11.        if (l - i < i)
12.          optimisticSufxDistance = bestsubstring[l - i]
13.        else
14.          optimisticSufxDistance = 0;
15.        if optimisticPrefixDistance + optimisticSufxDistance  $\geq$  bestDistance
16.          (s, i) = Bypass(s, i, l, 4)
17.        else
18.          (s, i) = NextVertex(s, i, l, 4)
19.      else
20.        word = nucleotide string corresponding to ( $s_1, s_2, s_3, \dots, s_l$ )
21.        if TotalDistance(word, DNA) < bestDistance
22.          bestDistance = TotalDistance(word, DNA)
23.          bestWord = word
24.          (s, i) = NextVertex(s, i, l, 4)
25.   return bestWord
```

# CONSENSUS: Greedy Motif Search

- Find two closest  $l$ -mers in sequences 1 and 2 and forms  $2 \times l$  alignment matrix with  $\text{Score}(\mathbf{s}, 2, \text{DNA})$
- At each of the following  $t-2$  iterations CONSENSUS finds a “best”  $l$ -mer in sequence  $i$  from the perspective of the already constructed  $(i-1) \times l$  alignment matrix for the first  $(i-1)$  sequences
- In other words, it finds an  $l$ -mer in sequence  $i$  maximizing

$$\text{Score}(\mathbf{s}, i, \text{DNA})$$

under the assumption that the first  $(i-1)$   $l$ -mers have been already chosen

- CONSENSUS sacrifices optimal solution for speed: in fact the bulk of the time is actually spent locating the first 2  $l$ -mers



# Some Motif Finding Programs

- **CONSENSUS**  
*Hertz, Stromo (1989)*
- **GibbsDNA**  
*Lawrence et al (1993)*
- **MEME**  
*Bailey, Elkan (1995)*
- **RandomProjections**  
*Buhler, Tompa (2002)*
- **MULTIPROFILER**  
*Keich, Pevzner (2002)*
- **MITRA**  
*Eskin, Pevzner (2002)*
- **Pattern Branching**  
*Price, Pevzner (2003)*
- **PRUNER II**  
Ravi Vijaya Satya and  
A. Mukherjee (2005)