Algorithms for Bioinformatics Autumn 2011

VELI MÄKINEN

HTTP://WWW.CS.HELSINKI.FI/EN/COURSES/582670/2011/S/K/1

Lecture 2

EXHAUSTIVE SEARCH AND MOTIF FINDING

These slides are edited from those at http://bix.ucsd.edu/bioalgorithms/slides.php

Random Sample

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Implanting Motif **AAAAAAAAGGGGGGGG**

atgaccgggatactgatAAAAAAAAGGGGGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataAAAAAAAAGGGGGGGa

tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag

gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa

Where is the Implanted Motif?

atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaggggggga

tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaaaaaaaaggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggcttatag

gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtaaaaaaaaggggggcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaaaaggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga

Implanting Motif **AAAAAAGGGGGGG** with Four Mutations

Where is the Motif???

atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

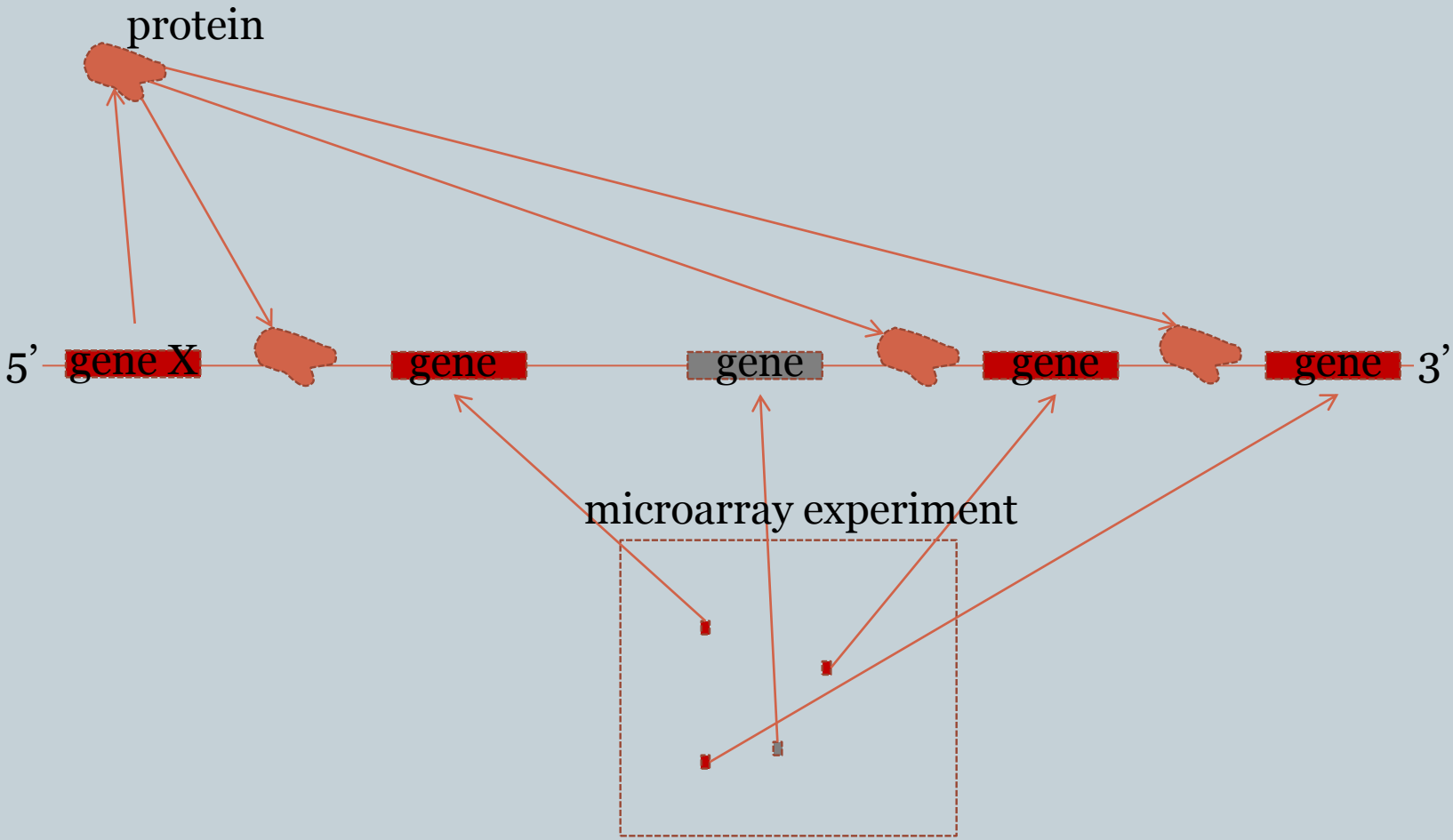
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga

Why Finding (15,4) Motif is Difficult?

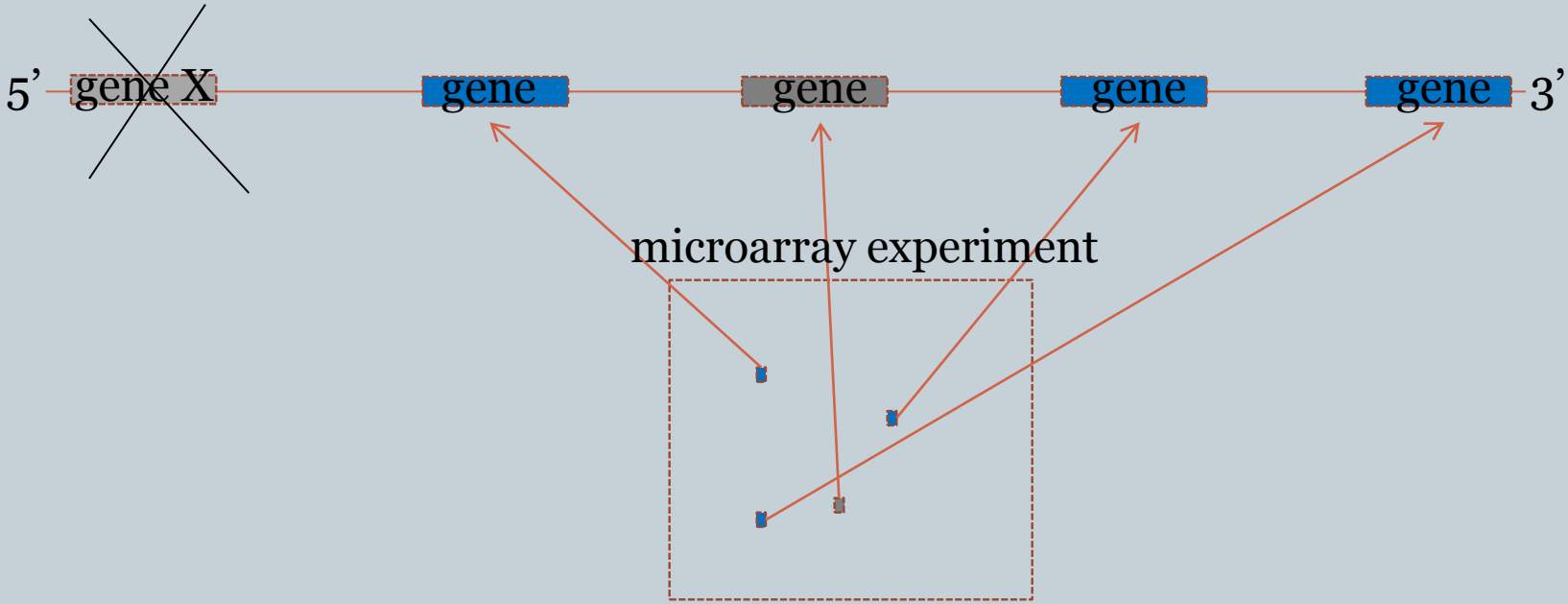
Challenge Problem

- 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.

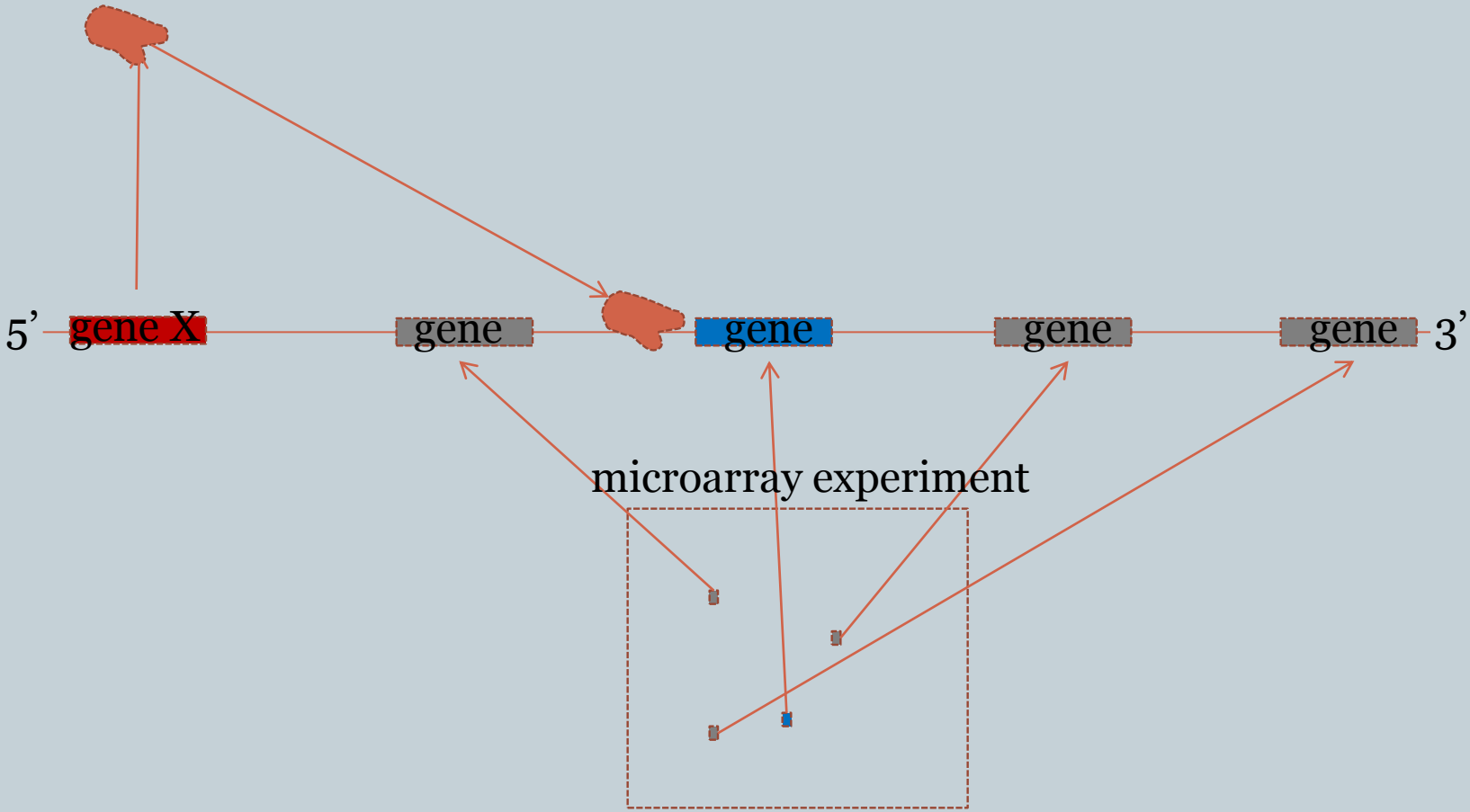
Biological motivation



Biological motivation cont'd



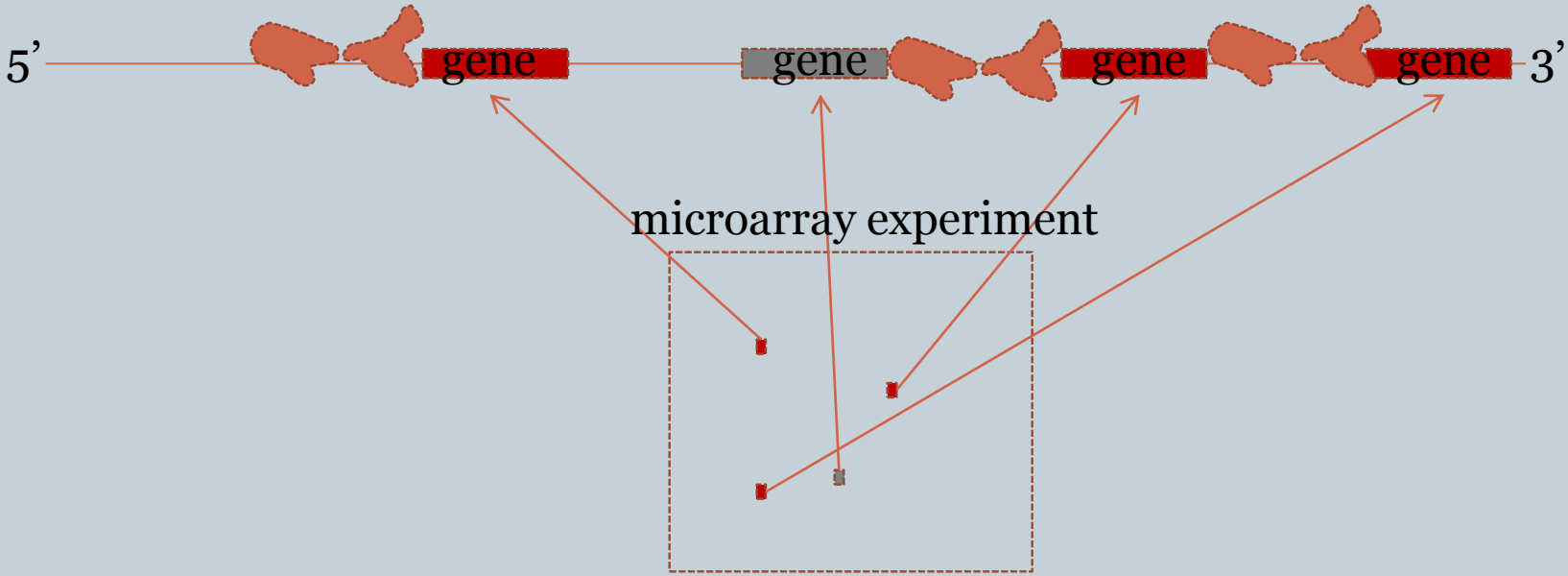
Or?



Or?

= protein product of gene X

= protein product of gene Y



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?**

Regulatory Proteins

- Gene X encodes regulatory protein, a.k.a. a ***transcription factor*** (TF)
- The 20 unexpressed genes rely on gene X's TF to induce transcription
- A single TF may regulate multiple genes

Regulatory Regions

- Every gene contains a regulatory region (RR) typically stretching 100-1000 bp upstream of the transcriptional start site
- Located within the RR are the ***Transcription Factor Binding Sites*** (TFBS), also known as ***motifs***, specific for a given transcription factor
- TFs influence gene expression by binding to a specific location in the respective gene's regulatory region - TFBS

Transcription Factor Binding Sites

- A TFBS can be located anywhere within the Regulatory Region.

- TFBS may vary slightly across different regulatory regions since non-essential bases could mutate

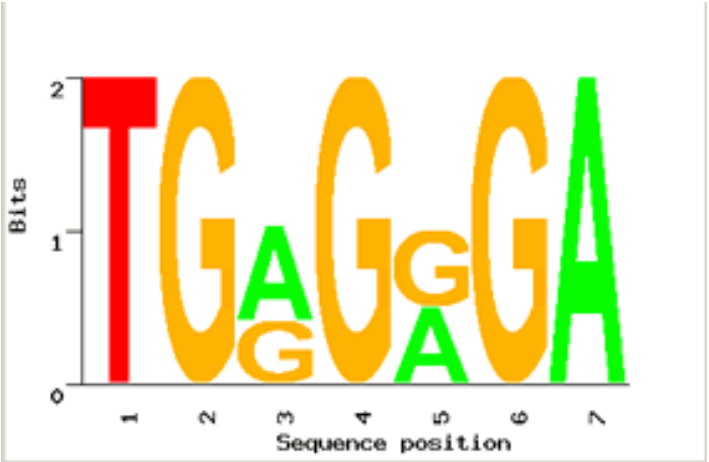
ACGATCGAGCTAGCTAGGCATGT

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
- Representations called *motif logos* illustrate the conserved and variable regions of a motif

TGGGGGA
TGAGAGA
TGGGGGA
TGAGAGA
TGAGGGA



Identifying Motifs

- 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 (TF) binds to a short DNA sequence called a motif (TFBS)
- 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?

The Motif Finding Problem

- Given a random sample of DNA sequences:

```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

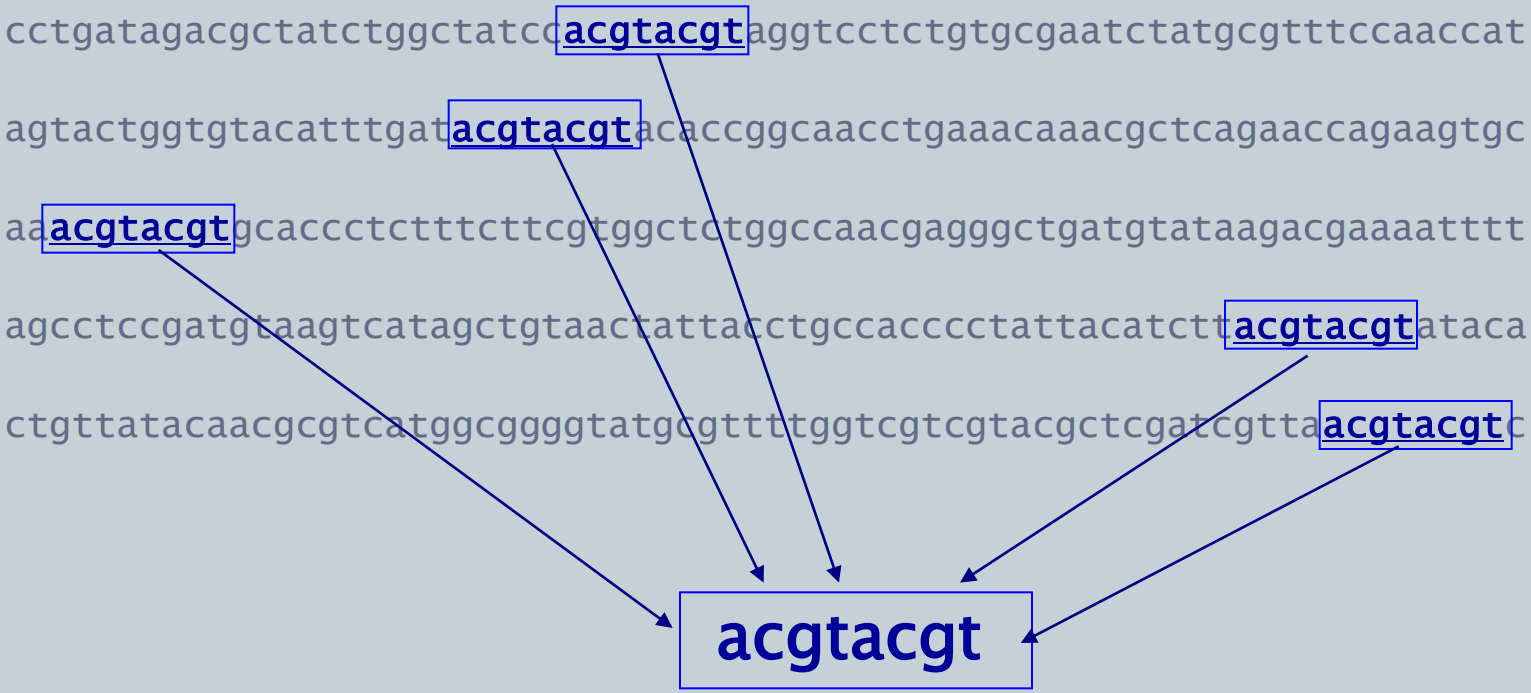
- Find the pattern that is implanted in each of the individual sequences, namely, the motif

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:



The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatcc**aGgtacTt**aggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgat**CcAtacgt**acaccggcaacctgaaacaaacgctcagaaccagaagtgc

aa**acgtTAgt**gcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatctt**acgtCcAt**ataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgtta**CcgtacgG**c

The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatcc**aGgtacTt**aggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgat**CcAtacgt**acaccggcaacctgaaacaaacgctcagaaccagaagtgc

aa**acgtTAgt**gcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

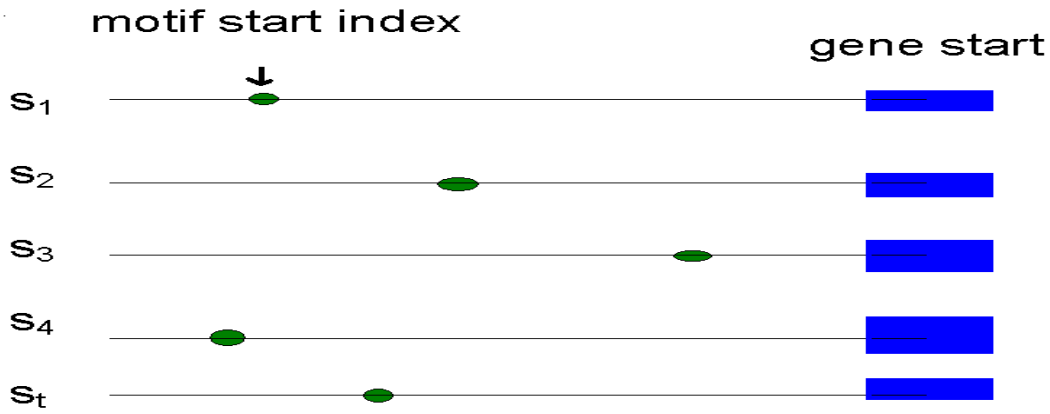
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatctt**acgtCcAt**ataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgtta**CcgtacgG**c

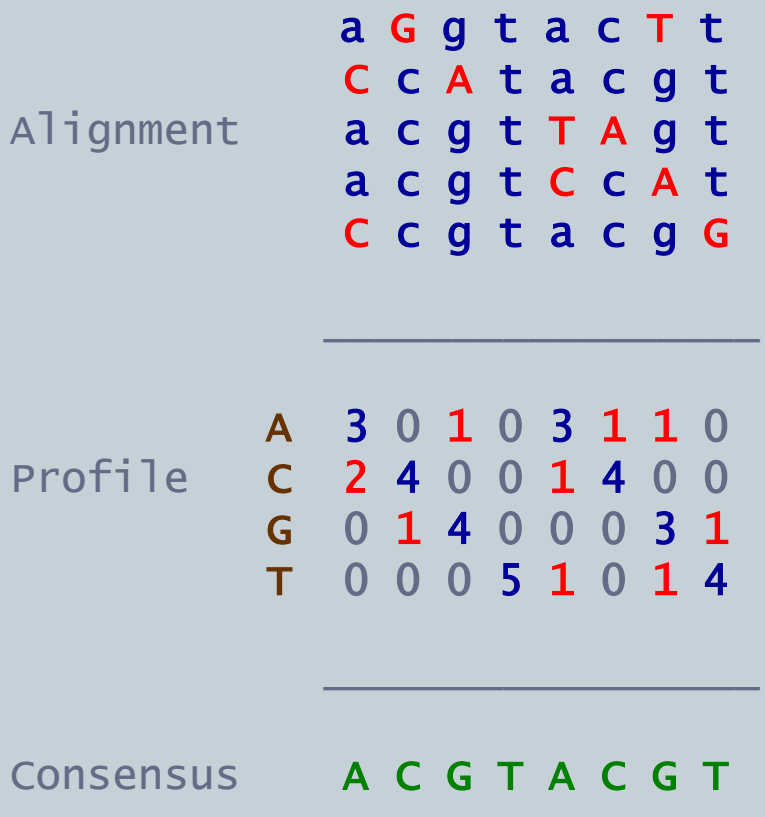
Can we still find the motif, now that we have 2 mutations?

Defining Motifs

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



Motifs: Profiles and Consensus



- Line up the patterns by their start indexes

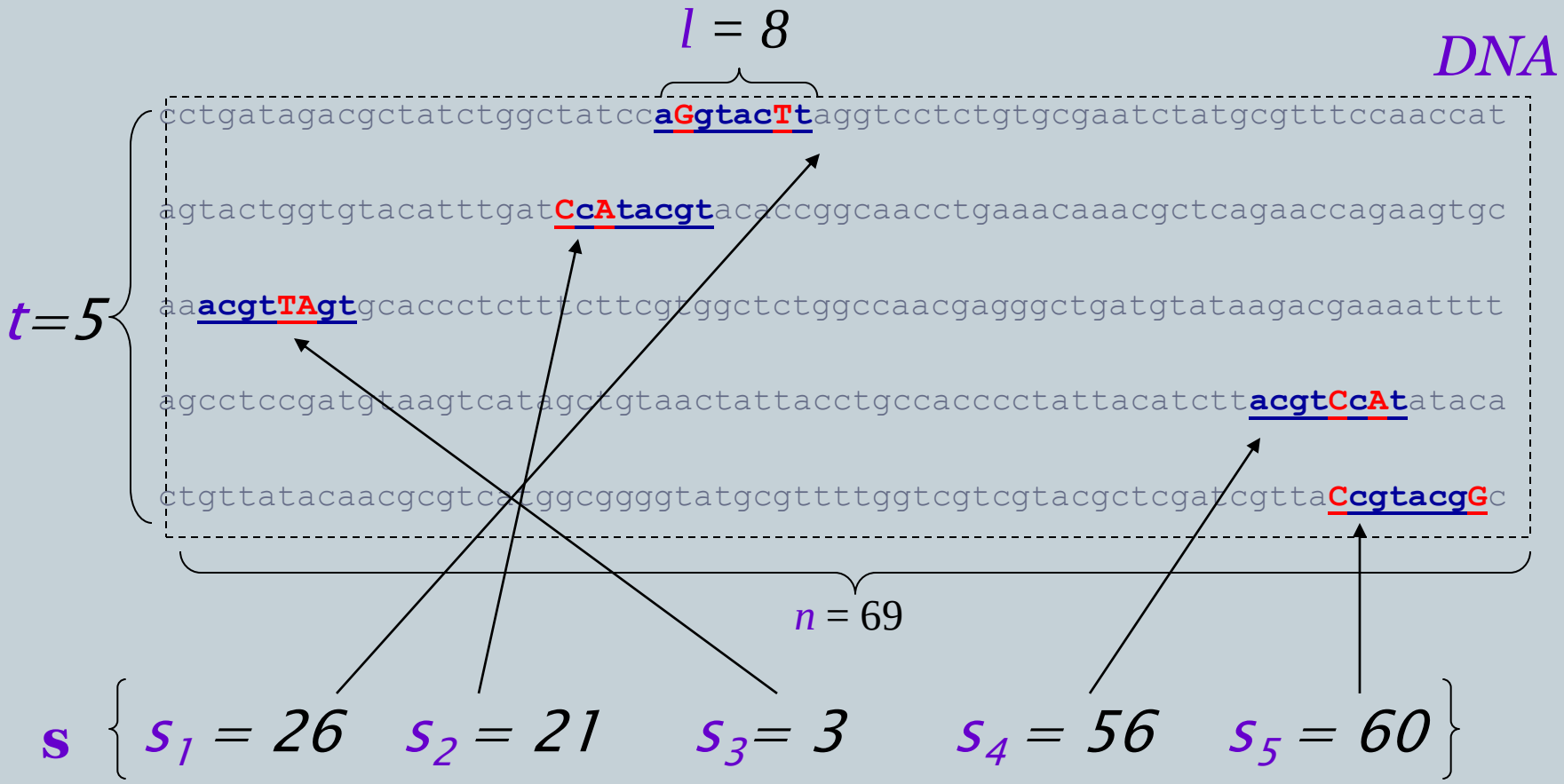
$$\mathbf{s} = (s_1, s_2, \ldots, s_t)$$

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

Defining Some Terms

- ***t*** - number of sample DNA sequences
- ***n*** - length of each DNA sequence
- ***DNA*** - sample of DNA sequences (*t* x *n* array)
  - DNA[i] denotes i-th DNA sequence.
  - DNA[i][j…j'] denotes substring T[j…j'], where T=DNA[i].
- ***l*** - length of the motif (*l*-mer)
- s_i - starting position of an *l*-mer in sequence *i*
- **s**=(s_1, s_2,… s_t) - array of motif's starting positions

Parameters


l = 8
DNA
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
t=5
n = 69
s
s1 = 26
s2 = 21
s3 = 3
s4 = 56
s5 = 60

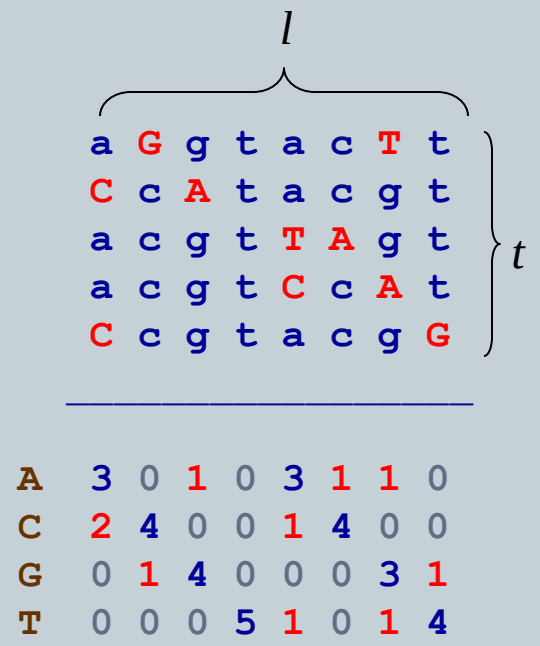
Scoring Motifs

- Given **s** = $(s_1, \ldots s_t)$ and ***DNA***:

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

where *count(c,j)* gives the number of times symbol *c* equals *DNA[i][s_i+j-1]*, that is,

$$count\ (c, j) = \left|\{(i \mid DNA\ [i][s_i + j - 1], i \in [1, t]\}\right|.$$



The Motif Finding Problem

- If starting positions **s**=(s_1, s_2,… 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 **s** 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 x n matrix of DNA, and l, the length of the pattern to find

- Output: An array of t starting positions $\mathbf{s} = (s_1, s_2, \ldots s_t)$ maximizing $Score(\mathbf{s}, DNA)$

The Motif Finding Problem: Brute Force Solution

- Compute the scores for each possible combination of starting positions **s**
- The best score will determine the best profile and the consensus pattern in ***DNA***
- The goal is to maximize *Score*(**s**,***DNA***) by varying the starting positions s_i, where:

$$s_i \in [1, \ldots, n-l+1]$$
$$i \in [1, \ldots, t]$$

BruteForceMotifSearch

1. BruteForceMotifSearch(*DNA, t, n, l*)
2. *bestScore* ← 0
3. **for** each **s**=(s_1,s_2 , . . ., s_t) from (1,1 . . . 1)
   to (*n*−*l*+1, . . ., *n*−*l*+1)
4. **if** (*Score*(**s**,*DNA*) > *bestScore*)
5. *bestScore* ← *score*(**s**, *DNA*)
6. **bestMotif** ← (s_1,s_2 , . . . , s_t)
7. **return bestMotif**

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 ℓt operations, so complexity is $\ell t(n - \ell + 1)^t = O(\ell t n^t)$

- 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

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

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

Total Distance: Example

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

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

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

Total Distance: Definition

- For each DNA sequence $\boldsymbol{i}$, compute all $d_H(\boldsymbol{v}, \boldsymbol{x})$, where $\boldsymbol{x}$ is an $\boldsymbol{l}$-mer with starting position s_i $(1 \leq s_i \leq \boldsymbol{n} - \boldsymbol{l} + 1)$
- Find minimum of $d_H(\boldsymbol{v}, \boldsymbol{x})$ among all $\boldsymbol{l}$-mers in sequence $\boldsymbol{i}$
- *TotalDistance(**v**,**DNA**)* is the sum of the minimum Hamming distances for each DNA sequence $\boldsymbol{i}$
- *TotalDistance(**v**,**DNA**)*=

$$\sum_{i \in [1,t]} \min_{s_i \in [1, n-l+1]} d_H(v, DNA[i][s_i \ldots s_i + l - 1])$$

The Median String Problem: Formulation

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

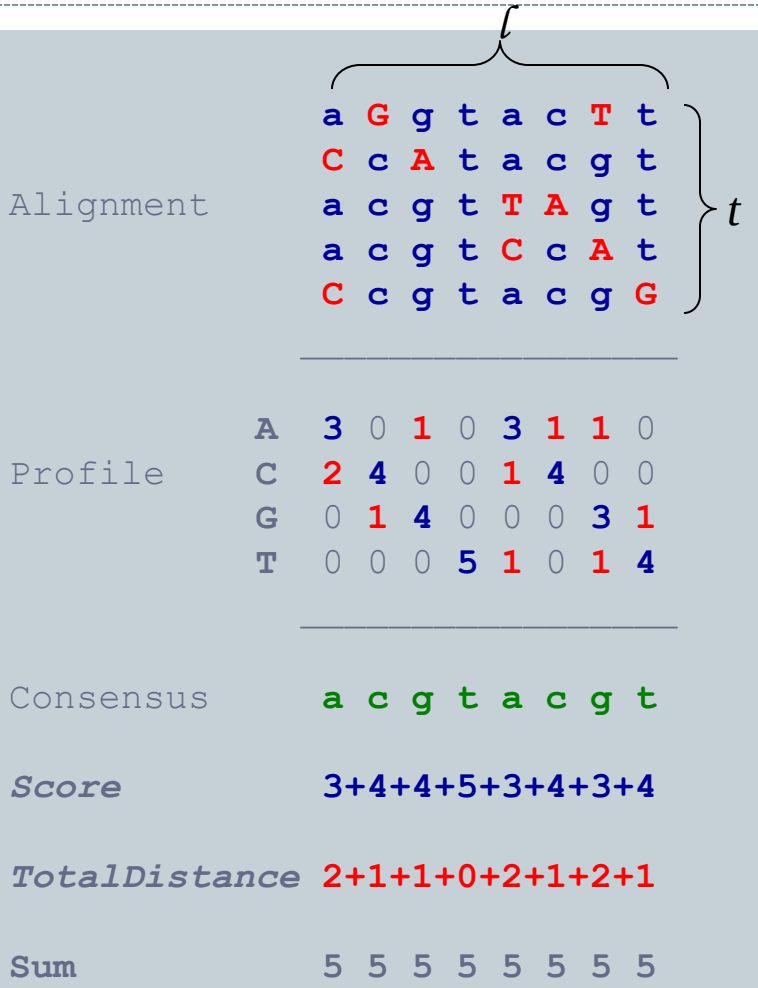
Median String Search Algorithm

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

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



- At any column i
  $Score_i + TotalDistance_i = t$
- Because there are ℓ columns
  $Score + TotalDistance = \ell * t$
- Rearranging:
  $Score = \ell * t - TotalDistance$
- $\ell * t$ is constant so 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 **s**.
    - Total running time $O(\ell t n^t)$.
  - The Median String Problem needs to examine all 4^ℓ combinations for $\boldsymbol{v}$.
    - Total running time $O(\ell t n 4^\ell)$.

Motif Finding: Improving the Running Time

Recall the BruteForceMotifSearch:

```
1.  BruteForceMotifSearch(DNA, t, n, l)
2.  bestScore ← 0
3.  for each s=(s1,s2 , . . ., st) from (1,1 . . . 1) to (n–l+1, . . ., n–l+1)
4.          if (Score(s,DNA) > bestScore)
5.                  bestScore ← Score(s, DNA)
6.                  bestMotif ← (s1,s2 , . . . , st)
7.  return bestMotif
```

Structuring the Search

- How can we perform the line

for each **s**=(s_1,s_2 , . . ., s_t) from (1,1 . . . 1) to ($n-\ell+1$, . . ., $n-\ell+1$) ?

- 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*** ← AAA…A
3. ***bestDistance*** ← ∞
4. **for** each *l*-mer **v from** AAA…A to TTT…T
   if *TotalDistance*(v,*DNA*) < ***bestDistance***
5. ***bestDistance***←*TotalDistance*(**v**,***DNA***)
6. ***bestWord*** ← **v**
7. **return *bestWord***

Structuring the Search

- For the Median String Problem we need to consider all 4^{ℓ} possible ℓ-mers:

ℓ

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

How to organize this search?

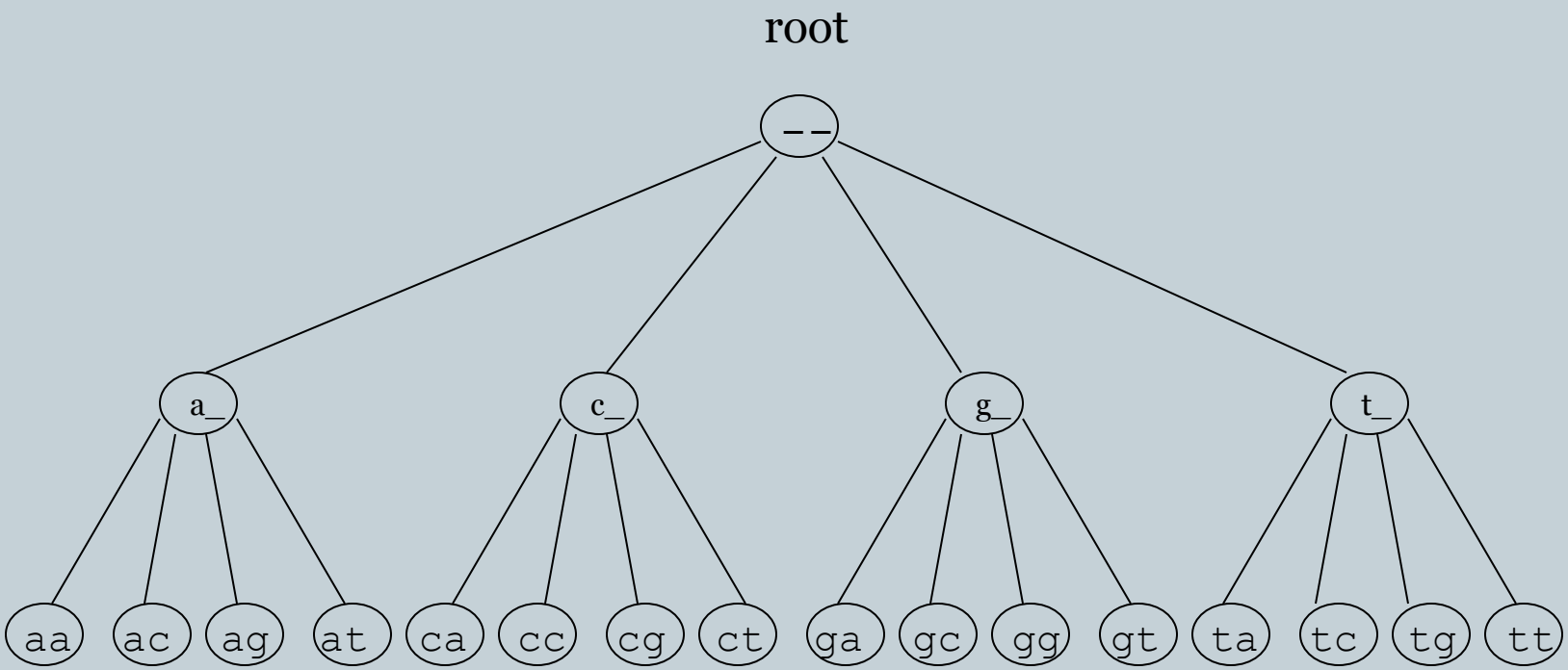
Alternative Representation of the Search Space

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

$$\overbrace{11\ldots11}^{l}$$
$$11\ldots12$$
$$11\ldots13$$
$$11\ldots14$$
$$\vdots$$
$$44\ldots44$$

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

Search 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
- How can we move through 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

Visit the Next Leaf

Given a current leaf **a** , we need to compute the "next" leaf:

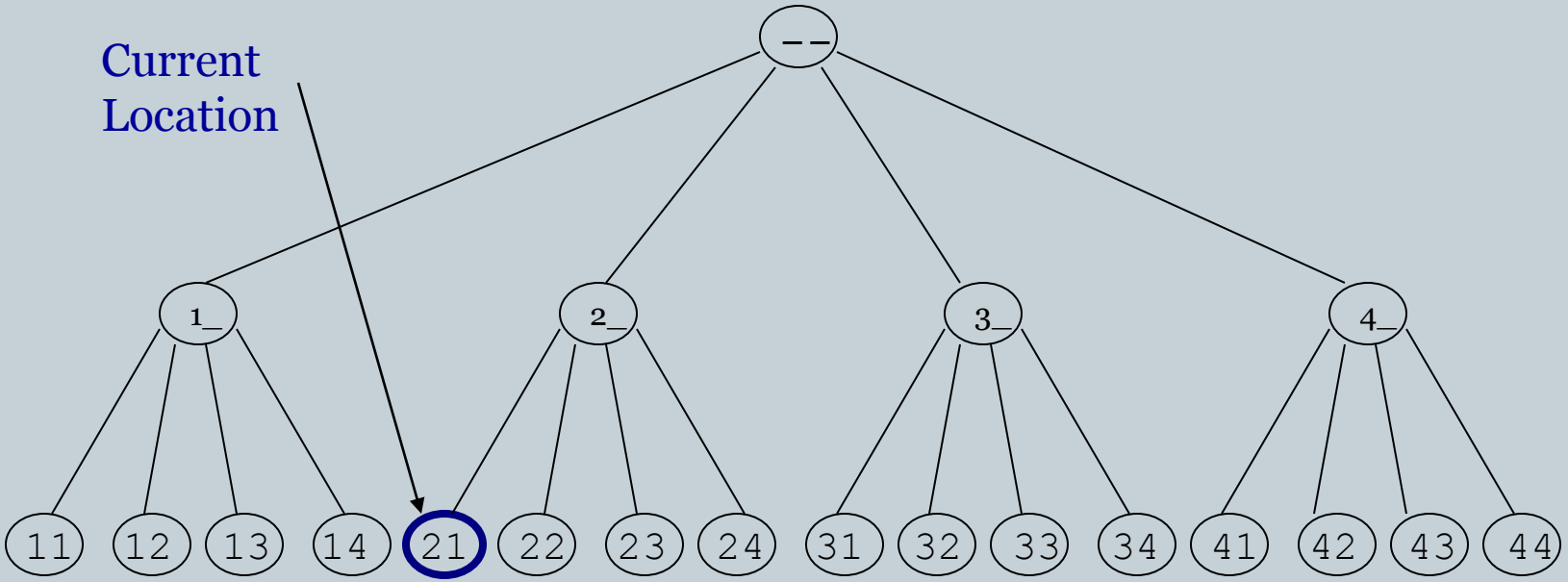
```
1.  NextLeaf( a,L, k )        // a : the array of digits
2.  for i ← L to 1            // L: length of the array
3.      if a_i < k            // k : max digit value
4.          a_i ← a_i + 1
5.          return a
6.      a_i ← 1
7.  return a
```

NextLeaf (cont'd)

- The algorithm is common addition in radix ***k*:**
  - Increment the least significant digit
  - "Carry the one" to the next digit position when the digit is at maximal value

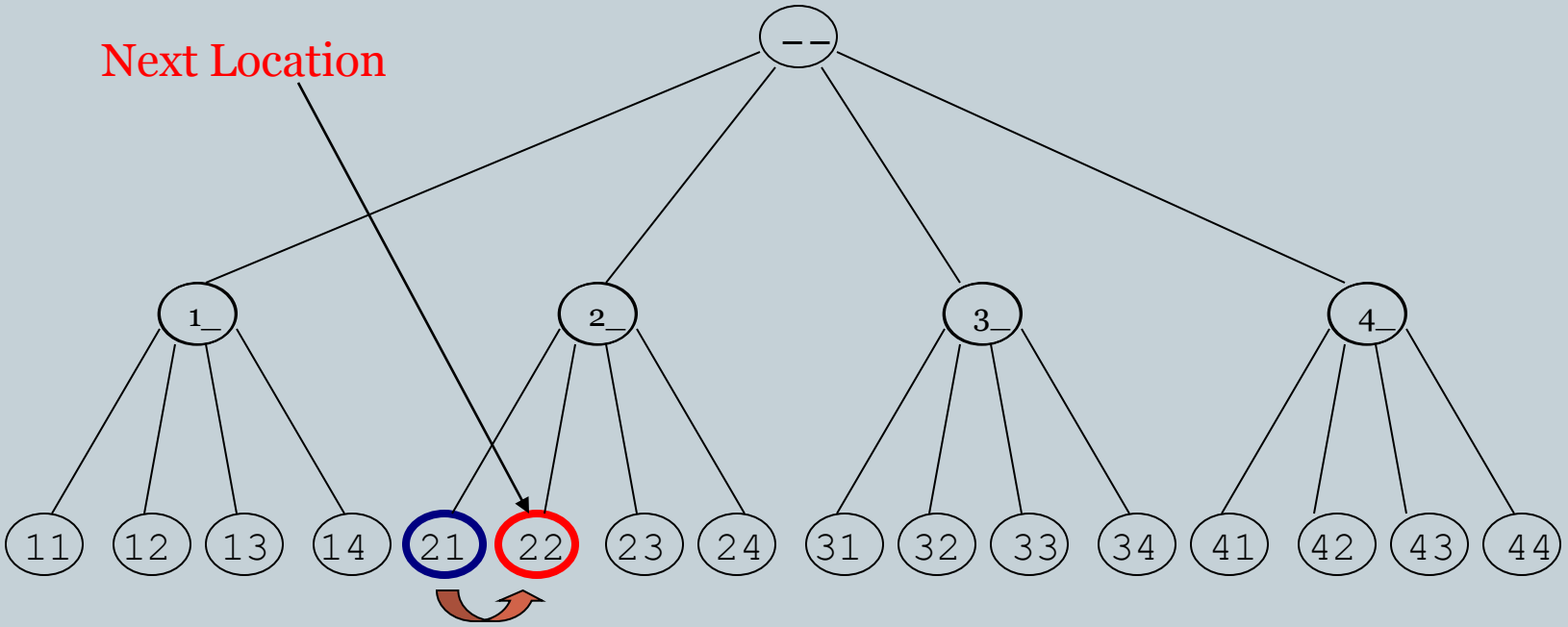
NextLeaf: Example

- Moving to the next leaf:



NextLeaf: Example (cont'd)

- Moving to the next leaf:



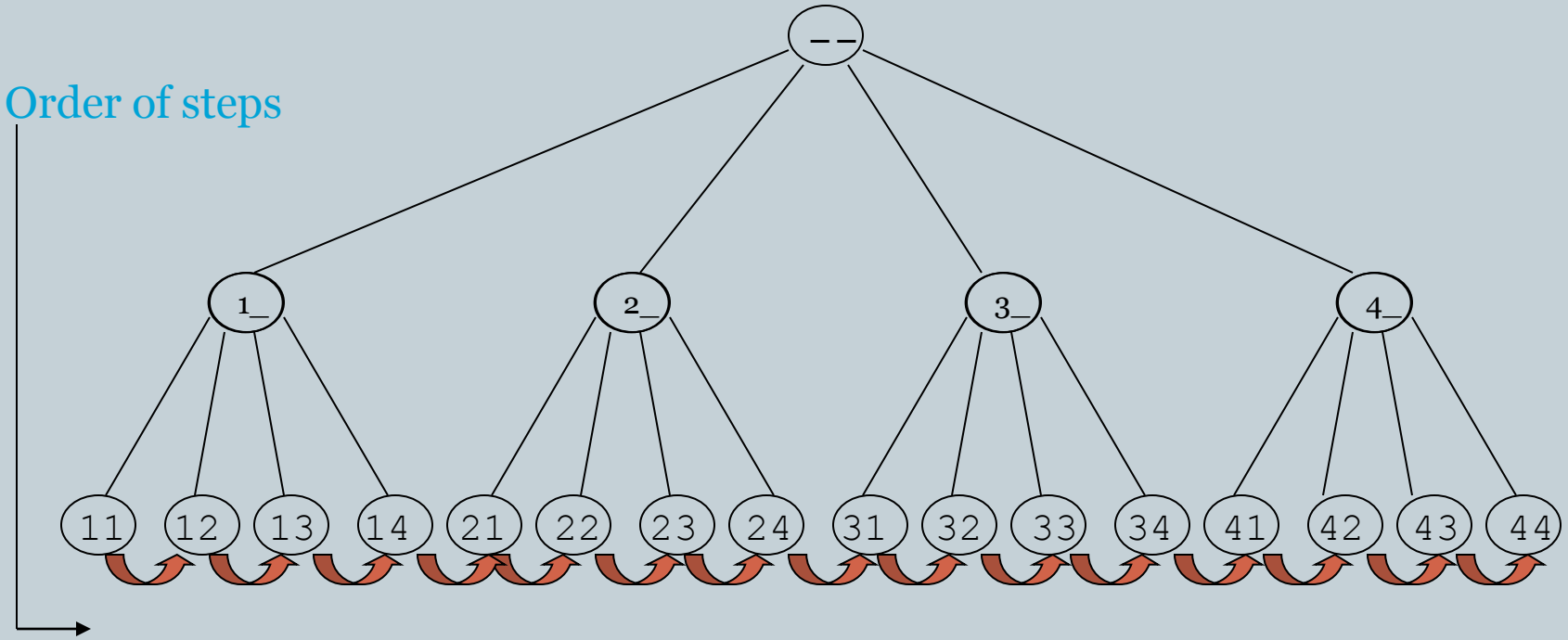
Visit All Leaves

- Printing all permutations in ascending order:

```
1.   AllLeaves(L,k)   // L: length of the sequence
2.   a ← (1,...,1)    // k : max digit value
3.   while forever    // a : array of digits
4.       output a
5.       a ← NextLeaf(a,L,k)
6.       if a = (1,...,1)
7.           return
```

Visit All Leaves: Example

- Moving through all the leaves in order:



Depth First Search

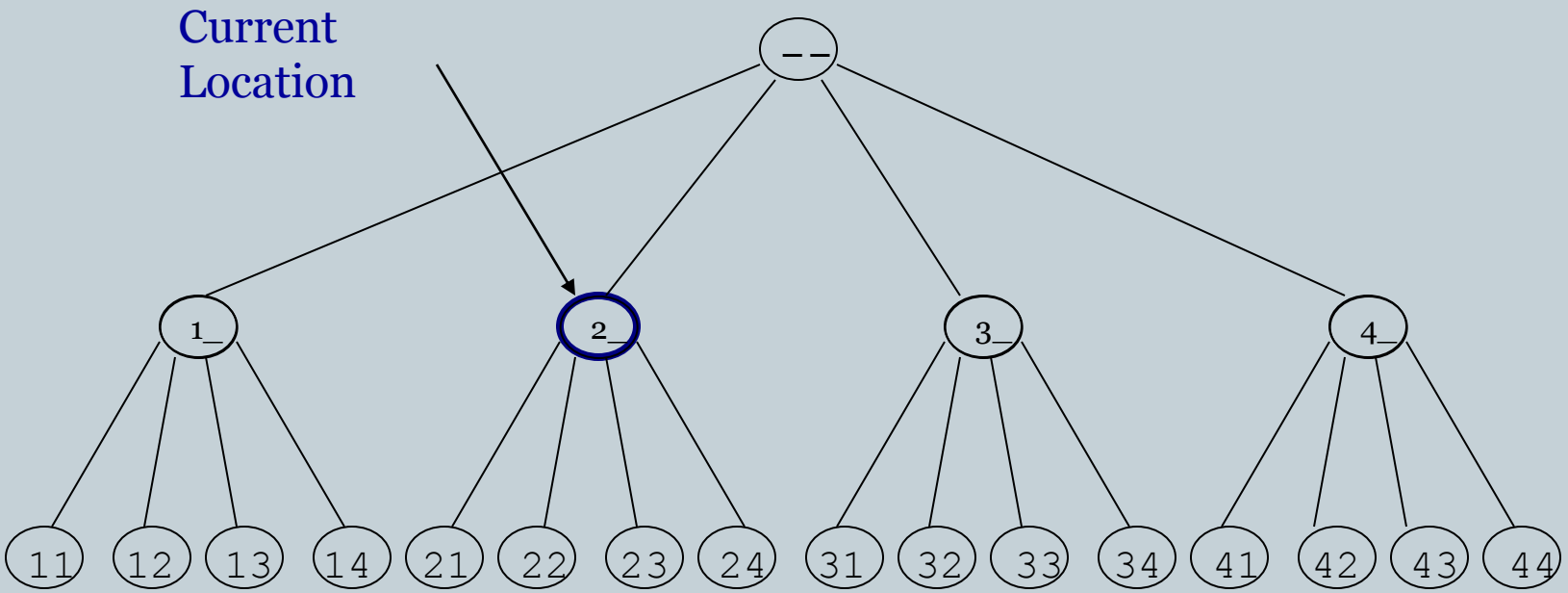
- So we can search leaves
- How about searching all vertices of the tree?
- We can do this with a *depth first* search

Visit the Next Vertex

```
1.  NextVertex(a,i,L,k)      // a : the array of digits
2.    if i < L               // i  : prefix length
3.      a_{i+1} ← 1          // L: max length
4.      return ( a,i+1)      // k : max digit value
5.    else
6.      for j ← L  to  1
7.        if a_j < k
8.          a_j ← a_j + 1
9.          return( a,j )
10.   return(a,0)
```

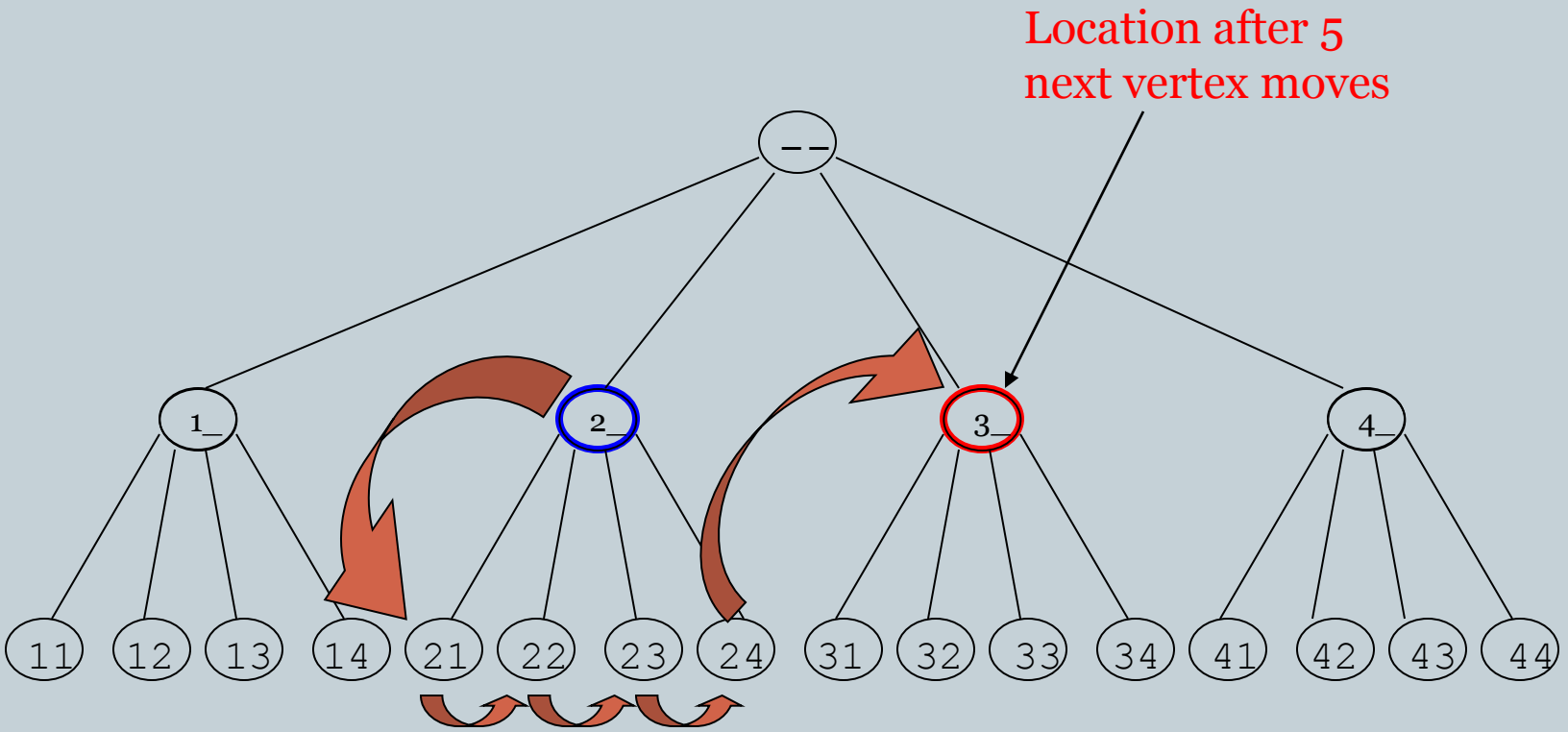
Example

- Moving to the next vertex:



Example

- Moving to the next vertices:



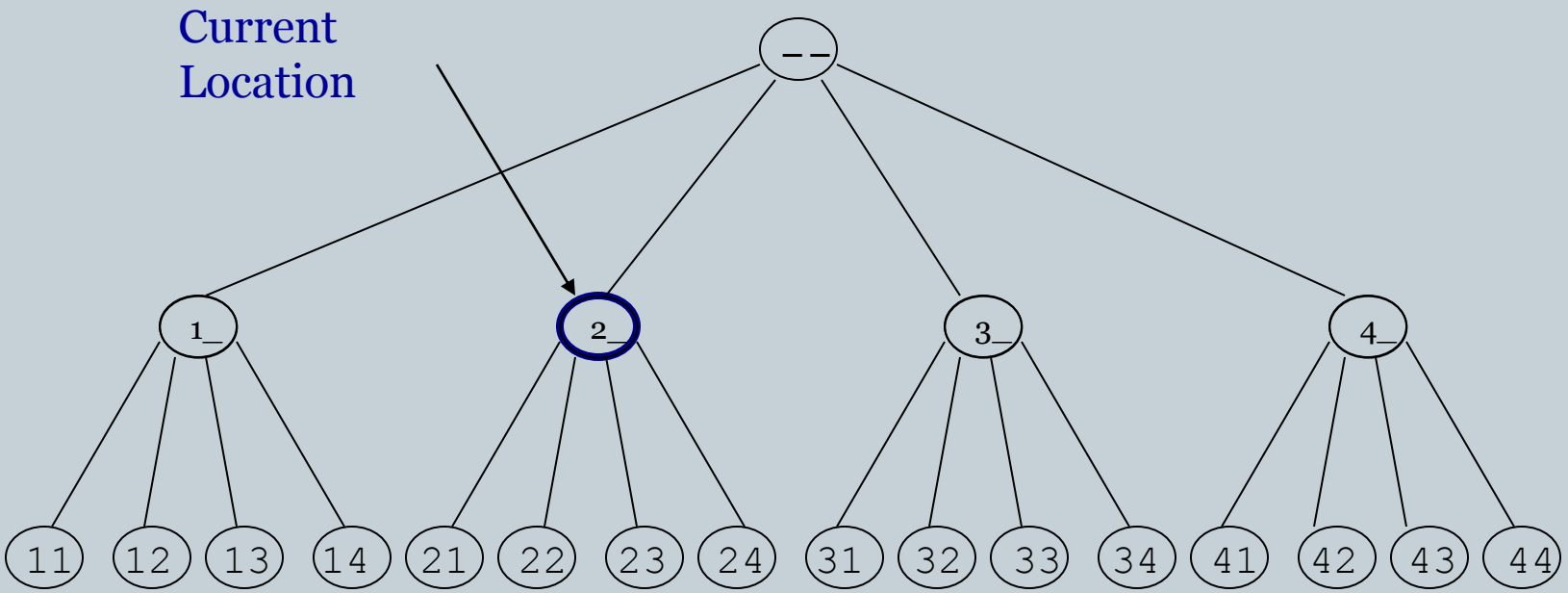
Bypass Move

- Given a prefix (internal vertex), find next vertex after skipping all its children

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

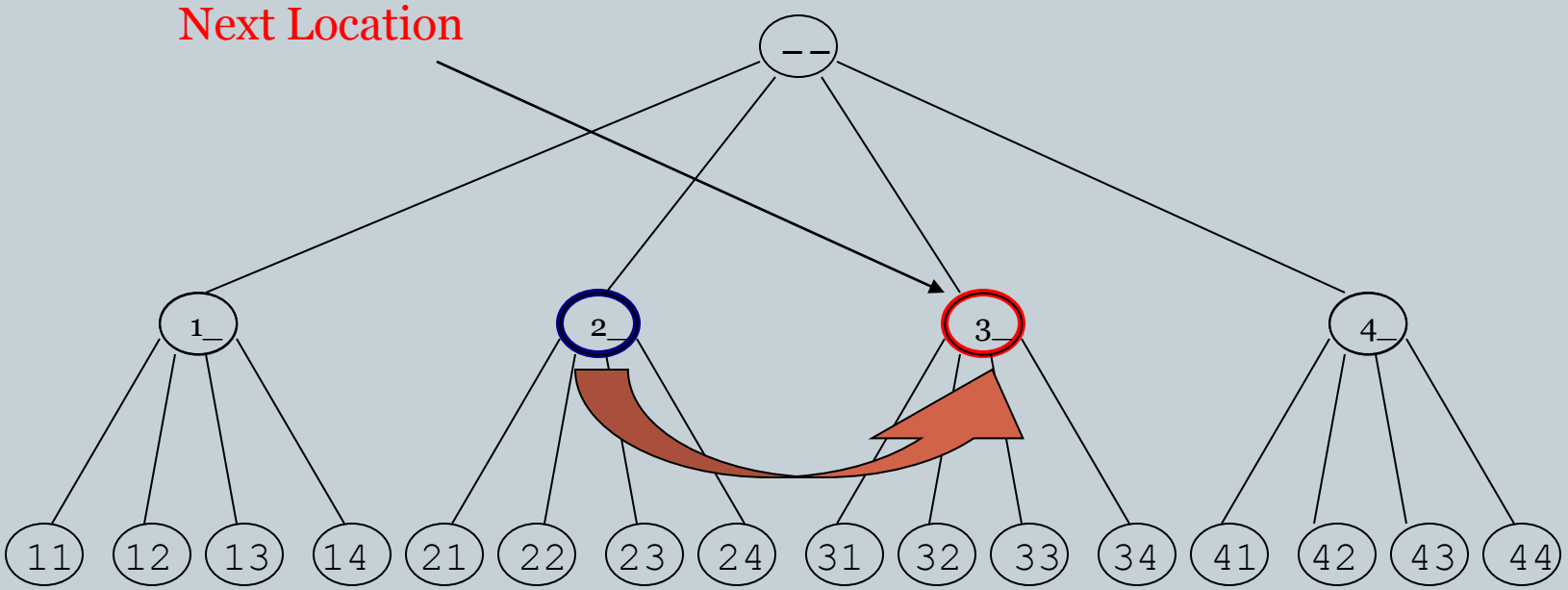
Bypass Move: Example

- Bypassing the descendants of "2-"



Example

- Bypassing the descendants of "2-"



Revisiting Brute Force Search

- Now that we have method for navigating the tree, lets look again at BruteForceMotifSearch

Brute Force Search Again

```
1.  BruteForceMotifSearchAgain(DNA, t, n, l)
2.  s ← (1,1,…, 1)
3.  bestScore ← Score(s,DNA)
4.  while forever
5.          s ←  NextLeaf (s, t, n– l + 1)
6.          if (Score(s,DNA) > bestScore)
7.                  bestScore ← Score(s, DNA)
8.                  bestMotif ← (s1,s2 , . . . , st)
9.  return bestMotif
```

Can We Do Better?

- Sets of $\mathbf{s}=(s_1, s_2, \ldots, s_t)$ may have a weak profile for the first i positions $(s_1, s_2, \ldots, s_i)$
- Every row of alignment may add at most ℓ to Score
- Define $Score(s, i, DNA) = \sum_{j=1}^{i} \max_{c \in \{A,T,C,G\}} count(c, j)$.
  - Recall definition of count() from the definition of Score().
- Optimism: if all subsequent $(t-i)$ positions $(s_{i+1}, \ldots s_t)$ add

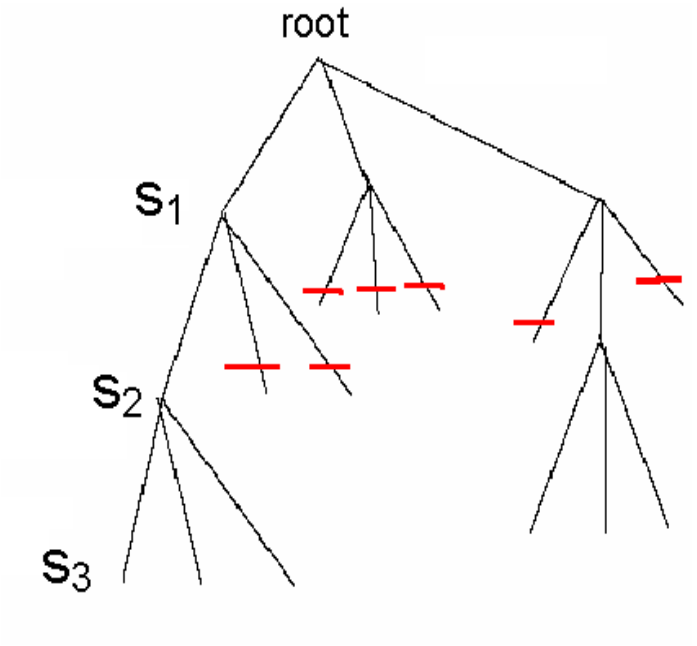
  $(t - i) * \ell$ to Score(**s**,*i*,*DNA*)

- If Score(**s**,*i*,*DNA*) + $(t - i) * \ell$ < **BestScore**, it makes no sense to search in vertices of the current subtree
  - Use **ByPass()**

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 $(n - l + 1)^{t-i}$ leaves
  - Use **NextVertex()** and **ByPass()** to navigate the tree



Pseudocode for Branch and Bound Motif Search

```
1.  BranchAndBoundMotifSearch(DNA,t,n,ℓ)
2.  s ← (1,...,1) # s_i=1 for i in [1,t]
3.  bestScore ← 0
4.  i ← 1
5.  while i > 0
6.      if i < t
7.          optimisticScore ← Score(s, i, DNA) +(t – i ) * ℓ
8.          if optimisticScore < bestScore
9.              (s, i) ← Bypass(s,i, n-ℓ+1)
10.         else
11.             (s, i) ← NextVertex(s, i, n-ℓ+1)
12.     else
13.         if Score(s,DNA) > bestScore
14.             bestScore ← Score(s)
15.             bestMotif ← (s_1, s_2, s_3, ..., s_t)
16.         (s,i) ← NextVertex(s,i,t,n-ℓ+ 1)
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 $(n-\ell+1)^t$ combinations for **s**.
  - The Median String Problem needs to examine 4^ℓ combinations of v. 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:

  TotalDistance (*prefix*, *DNA*) > *BestDistance*

  there is no use exploring the remaining part of the word

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

Bounded Median String Search

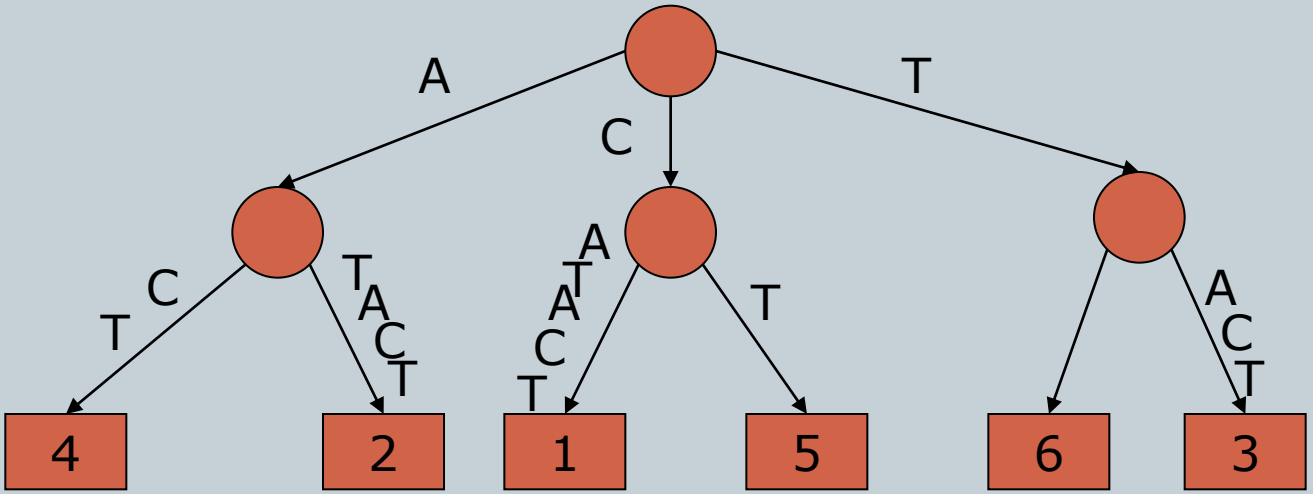
```
1.  BranchAndBoundMedianStringSearch(DNA,t,n,ℓ)
2.  v ← (1,…,1) # v_i=1 for i in [1,ℓ], recall: 1=A, 2=C, 3=G, 4=T
3.  bestDistance ← ∞
4.   i ← 1
5.  while i > 0
6.    if i < ℓ
7.       prefix ← string corresponding to the first i nucleotides of v
8.       optimisticDistance ← TotalDistance(prefix,DNA)
9.       if optimisticDistance > bestDistance
10.                  (v, i ) ← Bypass(v,i, ℓ, 4)
11.      else
12.                  (v, i ) ← NextVertex(v, i, ℓ, 4)
13.   else
14.      word ← nucleotide string corresponding to v
15.      if  TotalDistance(word,DNA) < bestDistance
16.                  bestDistance ← TotalDistance(word, DNA)
17.                  bestWord ← word
18.      (v,i ) ← NextVertex(v,i,ℓ, 4)
19. return bestWord
```

Some Motif Finding Programs

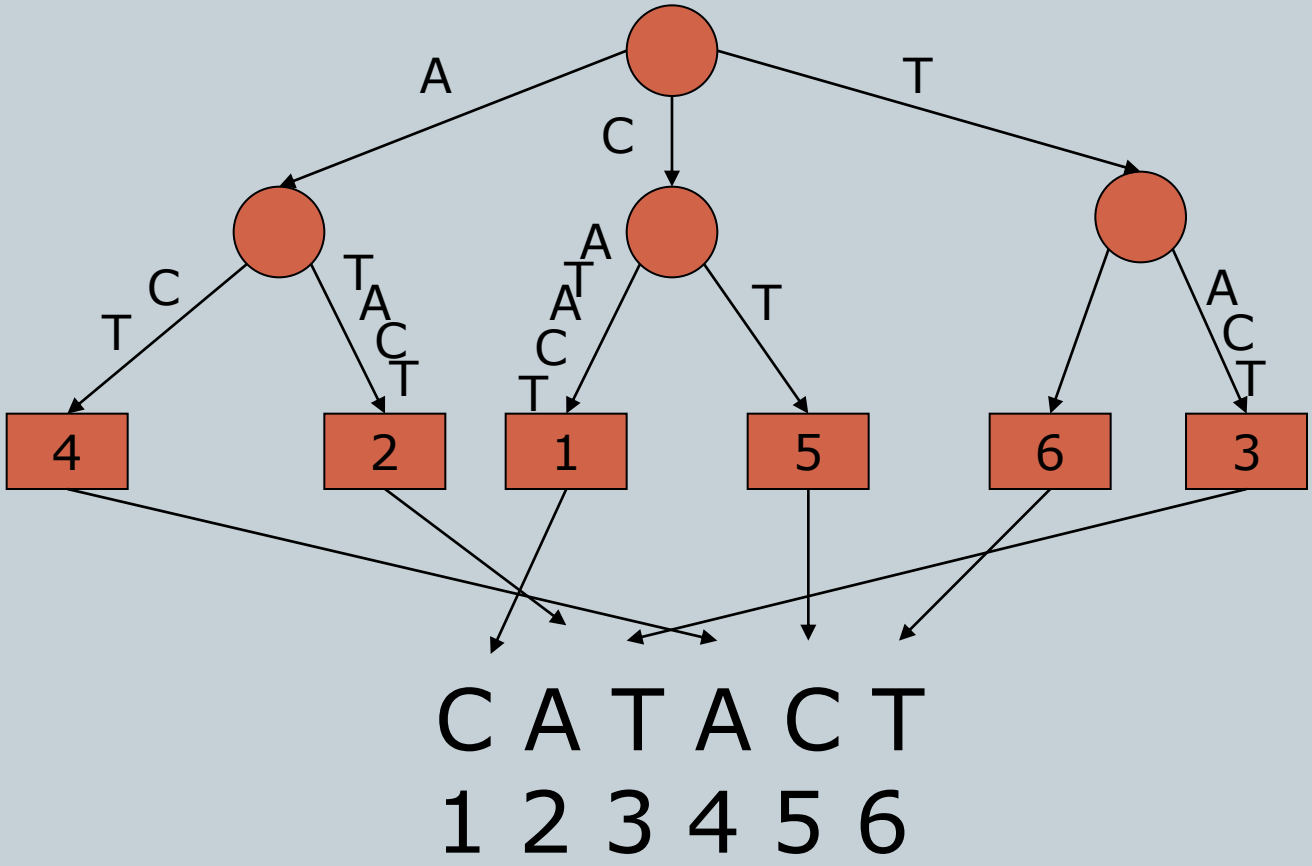
- **CONSENSUS**
  Hertz, Stromo (1989)
- **GibbsDNA**
  Lawrence et al (1993)
- **MEME**
  Bailey, Elkan (1995)
- ***Weeder***
  Pavesi, Mauri, Pesole (2001)
- **RandomProjections**
  Buhler, Tompa (2002)
- **MULTIPROFILER**
  Keich, Pevzner (2002)
- **MITRA**
  Eskin, Pevzner (2002)
- **Pattern Branching**
  Price, Pevzner (2003)

Weeder: a suffix tree –based approach

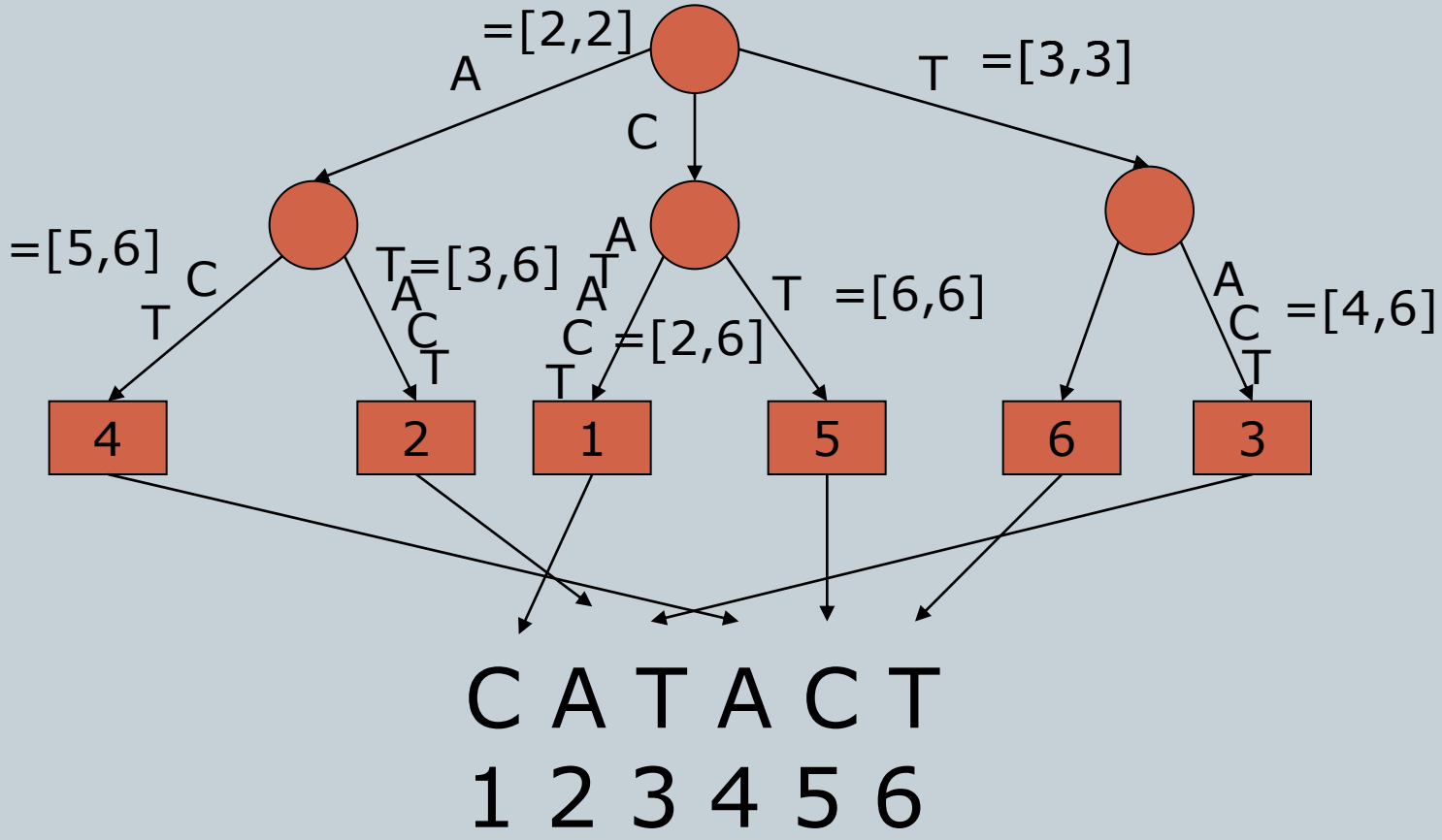
- Suffix tree is a compressed keyword trie of all *suffixes* of a sequence
- E.g. suffixes of sequence CATACT are CATACT, ATACT, TACT, ACT, CT, T.
  - suffix tree looks like:



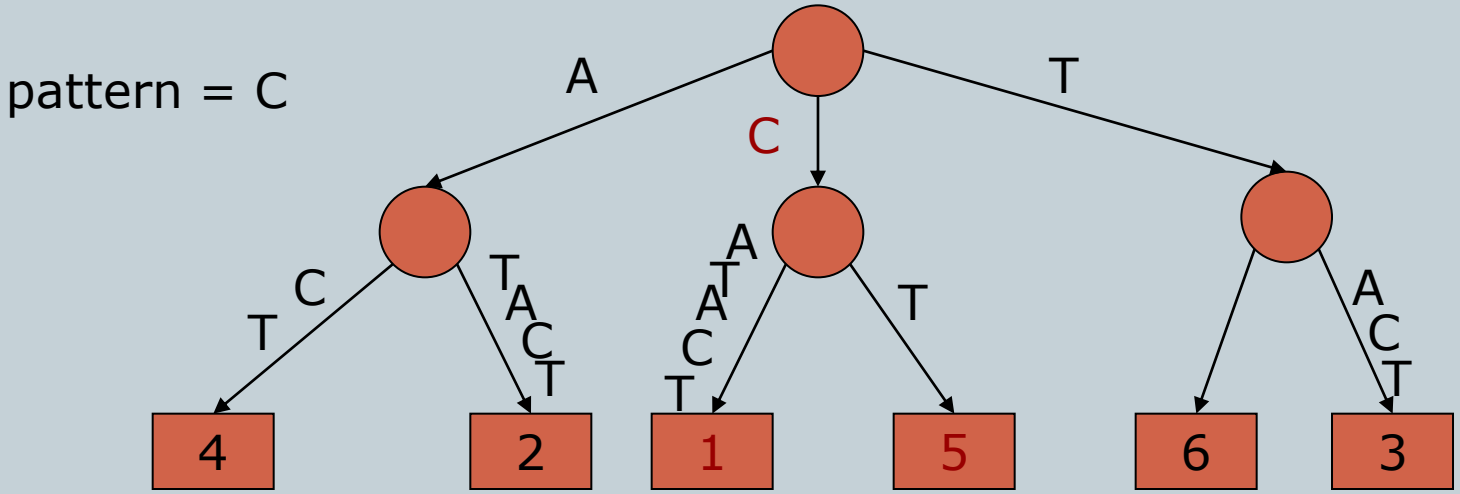
Suffix tree



Suffix tree



Exact search on suffix tree

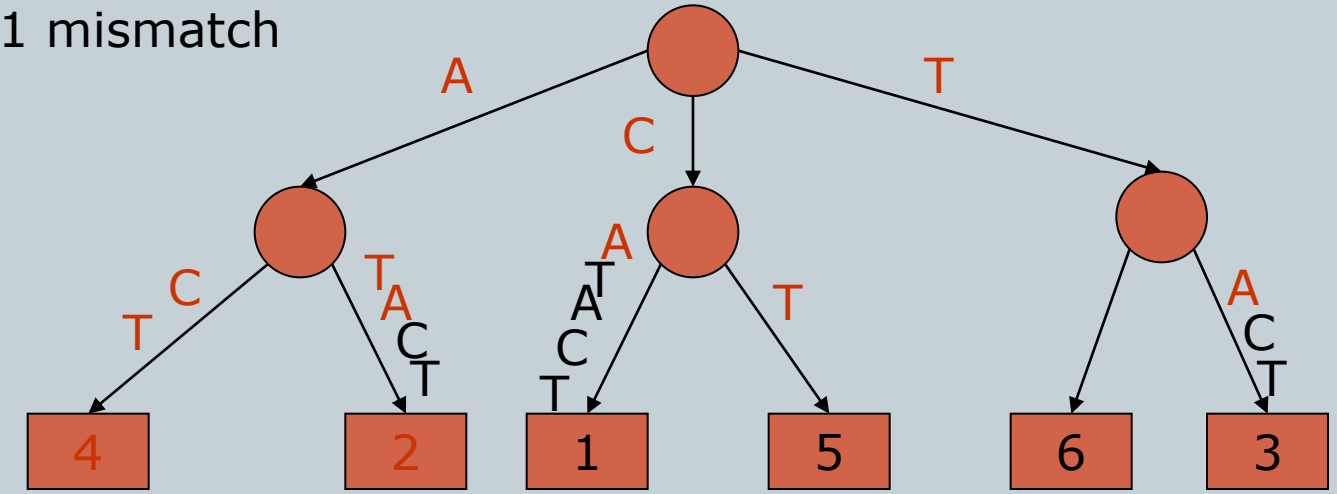


C A T A C T

1 2 3 4 5 6

Backtracking on suffix tree

ACA, 1 mismatch



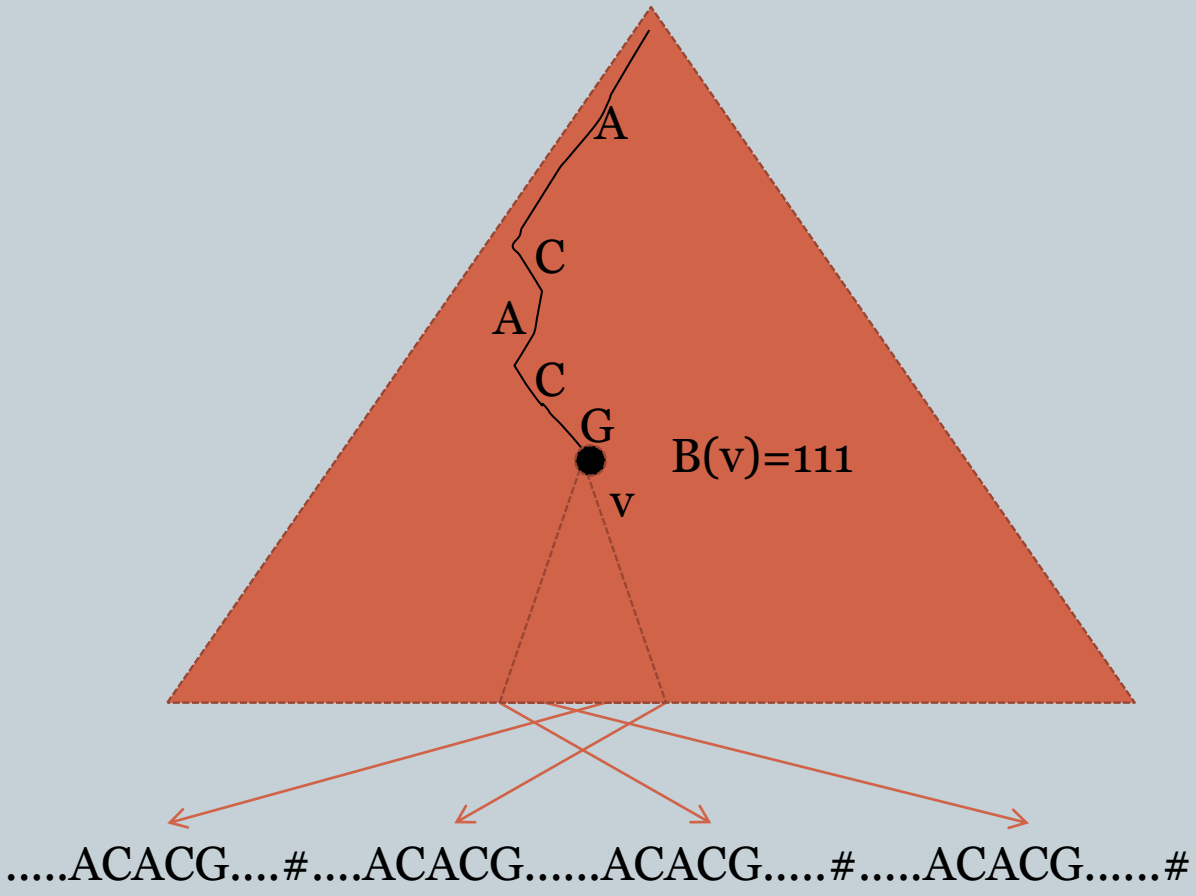
C A T A C T

1 2 3 4 5 6

Suffix tree and exact motif finding

- Concatenate the rows of the $t \times n$ matrix DNA into a string S, inserting an endmarker # between the rows.
- Build suffix tree T of S, and for each node v store a bitvector $B(v)[1,t]$ such that $B(v)[i]=1$ iff any suffix starting from row t is in the subtree of v.
- Let $v(s)$ denote an internal node of T such that the path from the root to the incoming edge of $v(s)$ spells s.
- String s is an *exact motif* if $B(v(s))[i]=1$ for all i.

Suffix tree and exact motif finding



Weeder: a suffix tree –based approach

- Weeder extends the exact motif finding algorithm to approximate motifs.
  - Backtracking plugged in.
  - Some heuristics to avoid too extensive branching.

Study group assignments

MONDAY 19.9. 12-14 B222

Group 1 (lastnames A-K)

- Read Sections 4.1-4.3 (partial digest problem) *before* coming to the study group meeting.
- Solve Problem 4.2 (page 119) *at* study group and use the solution to explain the material to the other groups:
  - Consider partial digest L={1,1,1,2,2,3,3,3,4,4,5,5,6,6,6,9,9,10,11,12,15}.
  - Solve the Partial Digest problem for L (i.e. find X such that $\Delta X = L$).

Group 2 (lastnames L-Sh)

- Read the following article *before* coming to the study group:
  - Alvis Brazma, Inge Jonassen, Jaak Vilo, and Esko Ukkonen. Predicting Gene Regulatory Elements in Silico on a Genomic Scale. *Genome Res*. 1998. 8: 1202-1215.
  - http://genome.cshlp.org/content/8/11/1202.full
  - Read especially section METHODS.
- *At* study group, discuss the approach, and draw a *pattern trie* for some small example input.

Group 3 (lastnames Si-Y)

- Read the following article *before* coming to the study group:
  - Pavesi G, Mauri G, Pesole G. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, 2001, 17(Suppl 1):S207-S214.
  - http://bioinformatics.oxfordjournals.org/content/17/suppl_1/S207.full.pdf
- *At* study group, summarize the message of the article and share the message to other groups.