Exhaustive search

Torgeir R. Hvidsten

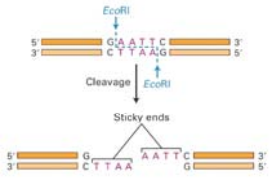
This lecture

- Restriction enzymes and the partial digest problem
- Finding regulatory motifs in DNA Sequences
- Exhaustive search methods

Restriction enzymes and the partial digest problem

Restriction enzymes

- *Hin*dII - first restriction enzyme – was discovered accidentally in 1970 while studying how the bacterium *Haemophilus influenzae* takes up DNA from the virus
- Restriction enzymes are used as a defense mechanism by bacteria to break down the DNA of attacking viruses

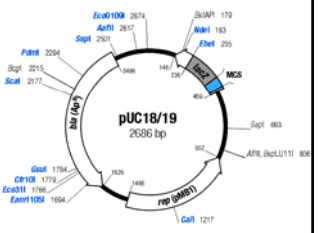


Uses of restriction enzymes

- Recombinant DNA technology (i.e. combining DNA sequences that would not normally occur together)
- Cloning
- cDNA/genomic library construction
- DNA mapping

Restriction maps

- A map showing the positions of restriction sites in a DNA sequence
- If the DNA sequence is known then constructing a restriction map is trivial
- In early days of molecular biology, DNA sequences were often unknown
- Biologists had to solve the problem of constructing restriction maps without knowing DNA sequences

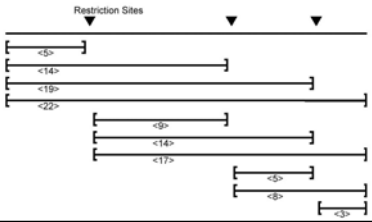


Measuring length of restriction fragments

- Restriction enzymes break DNA into restriction fragments
- Gel electrophoresis is a process for separating DNA by size and measuring sizes of restriction fragments
- Can separate DNA fragments that differ in length in only 1 nucleotide for fragments up to 500 nucleotides long

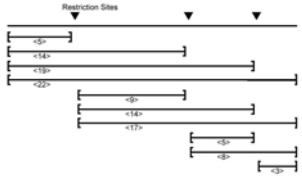
Partial restriction digest

- The sample of DNA is exposed to the restriction enzyme for only a limited amount of time to prevent it from being cut at all restriction sites
- This experiment generates the set of all possible restriction fragments between every two (not necessarily consecutive) cuts
- This set of fragment sizes is used to determine the positions of the restriction sites in the DNA sequence



Multiset of restriction fragments

- We assume that multiplicity of a fragment can be detected, i.e., the number of restriction fragments of the same length can be determined
- Restriction sites: $X = \{0, 5, 14, 19, 22\}$
- Multiset: $\Delta X = \{3, 5, 5, 8, 9, 14, 14, 17, 19, 22\}$



Partial Digest Problem (PDP)

X the set of n integers representing the location of all cuts in the restriction map, including the start and end *i.e.* $X = \{x_1 = 0, x_2, \ldots, x_n\}$.

n the total number of cuts

ΔX the multiset of integers representing lengths of each of the $n(n-1)/2$ fragments produced from a partial digest i.e. $\Delta X = \{x_j - x_i \mid 1 \le i < j \le n\}$

Problem: Given the multiset L, find a set X such that $\Delta X = L$

Partial Digest: Multiple Solutions (I)

It is not always possible to uniquely reconstruct a set X based only on ΔX.

For example, the set

$X = \{0, 2, 5\}$

and

$(X + 10) = \{10, 12, 15\}$

both produce $\Delta X = \{2, 3, 5\}$ as their partial digest set (they are homometric).

The sets $\{0,1,2,5,7,9,12\}$ and $\{0,1,5,7,8,10,12\}$ present a less trivial example of non-uniqueness. They both digest into:

$\{1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 6, 7, 7, 7, 8, 9, 10, 11, 12\}$.

Partial Digest: Multiple Solutions (II)

	0	1	2	5	7	9	12
0		1	2	5	7	9	12
1			1	4	6	8	11
2				3	5	7	10
5					2	4	7
7						2	5
9							3
12							

	0	1	5	7	8	10	12
0		1	5	7	8	10	12
1			4	6	7	9	11
5				2	3	5	7
7					1	3	5
8						2	4
10							2
12							

BruteForcePDP

```
BruteForcePDP(L, n)
1   M ← Maximum element in L
2   for every set of n – 2 integers 0 < x2 < … xn-1 < M
    such that xi ε L for 1 < i < n
3       X ← {0,x2,…,xn-1,M }
4       Form ΔX from X
5       if ΔX = L
6           return X
7   output "No solution"
```


Efficiency of BruteForcePDP

- The running time of BruteForcePDP is $O(|L|^{n-2})$ or, since $|L| = n(n-1)/2$, $O(n^{2n-4})$
- Note that without restricting the elements of X to elements in L, the running time would be $O(M^{n-2})$
- If $L = \{2, 998, 1000\}$ ($n = 3$, $M = 1000$), the running time would be very different
- Nonetheless, both algorithms have a exponential running time

Branch and Bound Algorithm for PDP

1. Initiation: Add the start (*0*) and end (*width*) point of the sequence to X (and remove the end point from L)
2. Find the largest element y in L
3. See if y fits on the right or left side of the restriction map by checking whether the other lengths (fragments) it creates are in L
4. If it fits, remove these lengths from L and add y (or *width* – y) to X (if not, backtrack)
5. Go back to step *2* until L is empty

Definitions

Before describing PartialDigest, first define

$\Delta(y, X)$

as the multiset of all distances between point y and all other points in the set X

$$\Delta(y, X) = \{|y - x_1|, |y - x_2|, \ldots, |y - x_n|\}$$

for $X = \{x_1, x_2, \ldots, x_n\}$

PartialDigest

```
PartialDigest(L)
1   width ← Maximum element in L
2   Remove width from L
3   X ← {0, width}
4   Place(L, X)

Place(L,X)
1   if L is empty
2       output X
3       return
4   y ← Maximum element in L
5   if Δ(y, X) ⊆ L
6       Remove lengths Δ(y, X) from L and add y to X
7       Place(L, X)
8       Remove y from X and add lengths Δ(y, X) to L
9   if Δ(width – y, X) ⊆ L
10      Remove lengths Δ(width – y, X) from L and add width – y to X and
11      Place(L, X)
12      Remove width – y from X and add lengths Δ(width – y, X) to L
13  return
```


An Example

$L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$

Initiation:
width = *10*, so delete *10* from L and add *0* and *10* to X

$L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$
$X = \{0, 10\}$

An Example

$L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$
$X = \{0, 10\}$

$y = 8$
$\sqcup(y, X) = \{8, 2\}$, which is a subset of L

$L = \{2, 3, 3, 4, 5, 6, 7\}$
$X = \{0, 8, 10\}$

An Example

$L = \{2, 3, 3, 4, 5, 6, 7\}$
$X = \{0, 8, 10\}$

$y = 7$
$\sqcup(y, X) = \{7, 1, 3\}$, which is not a subset of L
$\sqcup(width - y, X) = \{3, 5, 7\}$, which is a subset of L

$L = \{2, 3, 4, 6\}$
$X = \{0, 3, 8, 10\}$

An Example

$L = \{2, 3, 4, 6\}$
$X = \{0, 3, 8, 10\}$

$y = 6$
$\sqcup(y, X) = \{6, 3, 2, 4\}$, which is a subset of L

$L = \{\}$
$X = \{0, 3, 6, 8, 10\}$

An Example

$L = \{\}$
$X = \{0, 3, 6, 8, 10\}$

L is empty!
Output X

And continue searching for more solutions …

An Example

$L = \{2, 3, 4, 6\}$
$X = \{0, 3, 8, 10\}$

$y = 6$
$\sqcup(y, X) = \{6, 3, 2, 4\}$, which is a subset of L
$\sqcup(width - y, X) = \{4, 1, 4, 6\}$, which is not a subset of L

Backtrack!

An Example

$L = \{2, 3, 3, 4, 5, 6, 7\}$
$X = \{0, 8, 10\}$

$y = 7$
$\sqcup(y, X) = \{7, 1, 3\}$, which is not a subset of L
$\sqcup(width - y, X) = \{3, 5, 7\}$, which is a subset of L

Backtrack!

An Example

$L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$
$X = \{0, 10\}$

$y = 8$
$\lfloor(y, X) = \{8, 2\}$, which is a subset of L
$\lfloor(width - y, X) = \{2, 8\}$, which is a subset of L

$L = \{2, 3, 3, 4, 5, 6, 7\}$
$X = \{0, 2, 10\}$

An Example

$L = \{2, 3, 3, 4, 5, 6, 7\}$
$X = \{0, 2, 10\}$

$y = 7$
$\lfloor(y, X) = \{7, 5, 3\}$, which is a subset of L

$L = \{2, 3, 4, 6\}$
$X = \{0, 2, 7, 10\}$

An Example

$L = \{2, 3, 4, 6\}$
$X = \{0, 2, 7, 10\}$

$y = 6$
$\lfloor(y, X) = \{6, 4, 1, 4\}$, which is not a subset of L
$\lfloor(width - y, X) = \{4, 2, 3, 6\}$, which is a subset of L

$L = \{\}$
$X = \{0, 2, 4, 7, 10\}$

An Example

$L = \{\}$
$X = \{0, 2, 4, 7, 10\}$

L is empty!
Output X

And continue searching for more solutions …

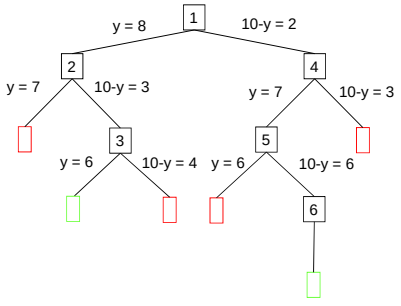
An Example

$L = \{2, 3, 3, 4, 5, 6, 7\}$
$X = \{0, 2, 10\}$

$y = 7$
$\lfloor(y, X) = \{7, 5, 3\}$, which is a subset of L
$\lfloor(width - y, X) = \{3, 1, 7\}$ which is not a subset of L

Backtrack and finish!

Recursion tree



Analyzing PartialDigest (I)

- Let $T(n)$ be the time PartialDigest takes to place n cuts
  - No branching case: $T(n) = T(n-1) + O(n)$
  - Branching case: $T(n) = 2T(n-1) + O(n)$
- The "No branching case" is quadratic $O(n^2)$ (like SelectionSort)
- The "Branching case" is exponential $O(2^n)$

Analyzing PartialDigest (II)

$T(n) = 2T(n-1) + O(n)$
$T(1) = 1$
$O(1) = 1$

$T(n) + O(n) = 2T(n-1) + O(n) + O(n) = 2(T(n-1) + O(n))$

Let $U(n) = T(n) + O(n)$

$U(n) = 2U(n-1)$
$U(1) = 2$

This gives the sequence: 2, 4, 8, 16, 32, 64, …→ $U(n) = 2^n$
$T(n) = 2^n - O(n)$

Finding regulatory motifs in DNA sequences

Random sample

```
atgacccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca
tgagtatcccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag
gtcaatcatgttcttgtgaatggatttttaactgaggggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggccccgtactgatgggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaaggggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca
```


Implanting motif AAAAAAAGGGGGGG

```
atgaccgggatactgatAAAAAAAGGGGGGGgccgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataAAAAAAAGGGGGGGa
tgagtatcccctgggatgacttAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAGGGGGGGcttatag
gtcaatcatgttcttgtgaatggatttAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggccccgtAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAGGGGGGGaccgaaaggggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAGGGGGGGa
```

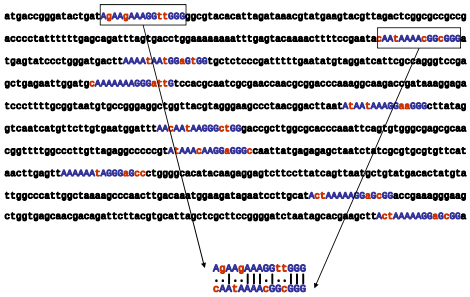

Where is the implanted motif?

```
atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaaggggggga
tgagtatcccctgggatgacttaaaaaaaaggggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgaaaaaaaaggggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaaggggggggcttatag
gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggccccgtaaaaaaaaggggggggcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttaaaaaaaaggggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaaggggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga
```

Implanting motif **AAAAAAGGGGGGG** with four random mutations

Where is the motif?

Why finding motif is difficult



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 a 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

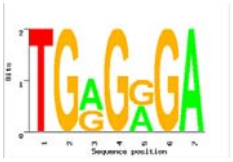
Regulatory regions

- Every gene contains a regulatory region (RR) typically stretching 100-1000 bp upstream of the transcriptional start site
- Located somewhere 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 - and recruiting the DNA polymerase

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



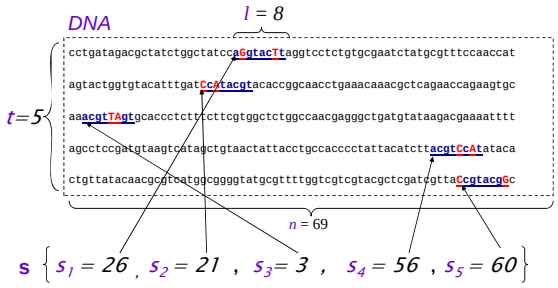
The motif finding problem

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

Parameters



Definitions

t	number of sample DNA sequences
n	length of each DNA sequence
DNA	sample of DNA sequences (t x n array)
l	length of the motif (l - mer)
s_i	starting position of an l-mer in sequence i
$\mathbf{s}=(s_1, s_2, \ldots s_t)$	array of motif starting positions

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
```

- Line up the patterns by their start indexes

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

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
```

- Construct matrix profile with frequencies of each nucleotide in columns

Consensus

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

- Consensus nucleotide in each position has the highest score in column

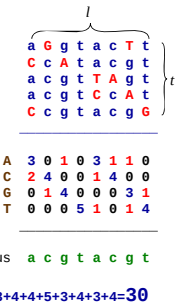
Consensus

- 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
- We need to introduce a scoring function to compare different motifs and choose the "best" one.

Scoring motifs: consensus score

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

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



The motif finding problem

- 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 ***DNA***, and l, the length of the pattern to find
- Output: An array of t starting positions $\boldsymbol{s} = (s_1, s_2, \ldots s_t)$ maximizing Score($\boldsymbol{s}$,***DNA***)

BruteForceMotifSearch

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

Running Time of BruteForceMotifSearch

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

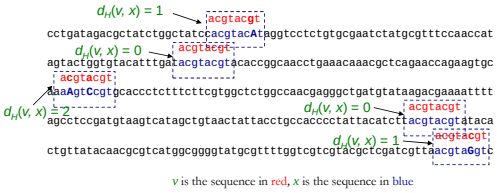
Hamming Distance

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

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

Total Distance: Example

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



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

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

The median string problem

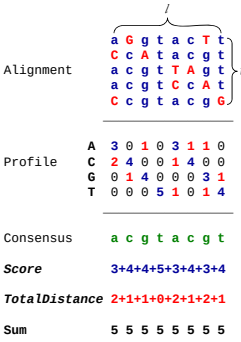
- Goal: Given a set of DNA sequences, find a median string
- Input: A $t \times n$ matrix ***DNA***, and l, the length of the pattern to find
- Output: A string v of l nucleotides that minimizes TotalDistance(v,***DNA***) over all strings of that length

Median string search algorithm

BruteForceMedianStringSearch (***DNA***, *t*, *n*, *l*)

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


Motif finding problem = median string problem



- At any column i $\text{Score}_i + \text{TotalDistance}_i = t$
- Because there are l columns Score + TotalDistance = $l \times t$
- Rearranging: Score = $l \times t$ - TotalDistance
- $l \times t$ is constant, thus the minimization of TotalDistance is equivalent to the maximization of Score

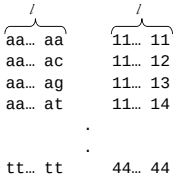
Motif finding problem vs. median string problem

Why bother reformulating the *motif finding* problem into the *median string* problem?

- The motif finding problem needs to examine all the combinations for $\boldsymbol{s}$. That is $(n - l + 1)^t$ combinations
- The median string problem needs only to examine all 4^l combinations for v.

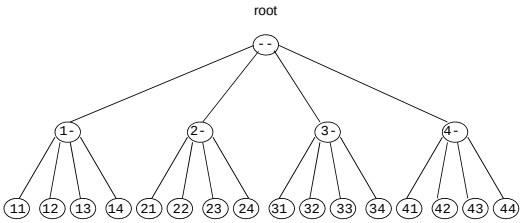
Structuring the search

For the median string problem we need to consider all 4^l possible l-mers:



How to organize this search?

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?

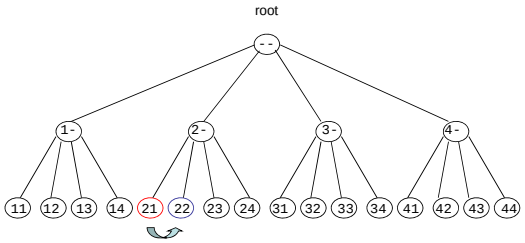
Visit the Next Leaf

Let $\boldsymbol{a}$ be an array of digits: $\boldsymbol{a} = \{a_1, a_2, \ldots a_L\}$ where $a_i \in [1,k]$, $1 \le i \le L$

Given a current leaf $\boldsymbol{a}$, we need to compute the "next" leaf:

```
NextLeaf(a,L,k)
1  for i ← L to 1
2    if a_i < k
3      a_i ← a_i + 1
4      return a
5    a_i ← 1
6  return a
```


NextLeaf: Example

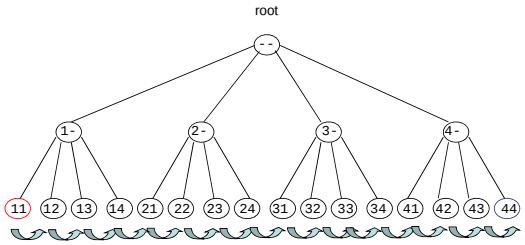


Visit all leaves

Printing all permutations in ascending order:

```
AllLeaves(L,k)
1  a ← (1,1,...,1)
2  while forever
3    output a
4    a ← NextLeaf(a,L,k)
5    if a = (1,1,...,1)
6      return
```


Visit all leaves: Example



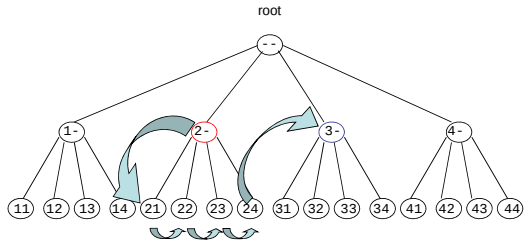
Visit the next vertex

We can search though all vertices of the tree with a *depth first* search

i is the prefix length

```
NextVertex(a,i,L,k)
1   if i < L
2      a_{i+1} ← 1
3      return (a,i+1)
4   else
5      for j ← L to 1
6         if a_j < k
7            a_j ← a_j + 1
8            return(a,j)
9   return(a,0)
```


Visit the next vertex: Example (five moves)

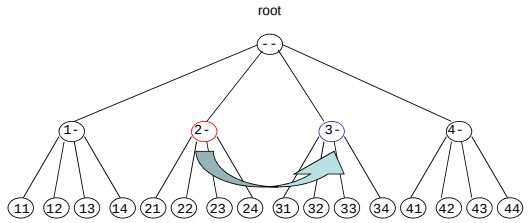


Bypass Move

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

```
Bypass(a,i,L,k)
1   for j ← i to 1
2      if a_j < k
3         a_j ← a_j + 1
4         return(a,j)
5   return(a,0)
```


Bypass Move: Example



Brute force search again

```
BruteForceMedianStringSearchAgain(DNA, t, n, l)
1   s ← (1,1,…, 1)
2   bestDistance ← ∞
3   while forever
4      s ← NextLeaf (s, l, 4)
5      word ← Nucleotide string corresponding to (s_1,s_2 , . . . , s_l)
6      if (TotalDistance(word,DNA) < bestDistance)
7         bestDistance ← TotalDistance(word, DNA)
8         bestWord ← word
9      if s = (1,1,…, 1)
10        return bestWord
```


Can We Do Better?

- Let TotalDistance(*prefix*,***DNA***) be the distance for a nucleotide string corresponding to (s_1, s_2, …,s_i)
- 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
- Use ByPass()!

Bounded Median String Search

```
BranchAndBoundMedianStringSearch(DNA,t,n,l)
1   s ← (1,…,1)
2   bestDistance ← ∞
3   i ← 1
4   while i > 0
5       if i < l
6           prefix ← Nucleotide string corresponding to (s1, s2, …, si)
7           optimisticDistance ← TotalDistance(prefix,DNA)
8           if optimisticDistance > bestDistance
9               (s, i) ← Bypass(s,i, l, 4)
10          else
11              (s, i) ← NextVertex(s, i, l, 4)
12      else
13          word ← Nucleotide string corresponding to (s1, s2, …, si)
14          if TotalDistance(s,DNA) < bestDistance
15              bestDistance ← TotalDistance(word, DNA)
16              bestWord ← word
17          (s,i) ← NextVertex(s,i,l, 4)
18  return bestWord
```


Running time

- As usual with branch-and-bound algorithms, there is no improved running time in the worst case
- However, it often results in a practical speedup