Algorithms and Data Structures in Biology

Exhaustive Search Algorithms

Ugo Dal Lago

University of Bologna, Academic Year 2018/2019

The Exhaustive Search Paradigm

- Exhaustive Search algorithms, also called *brute force algorithms*, are a sort of algorithms which:
  - Which typically have *high* (most often, exponential) complexity .
  - But which are often relatively easy to be proved correct.
- The idea behind an exhaustive search algorithm is that, whenever the problem *can be seen* as the problem of looking for an element in a finite set:
  - **Having** certain properties;
  - or being **the best** according to a given notion of optimality;
  - or cominations thereof.
- The complexity tend to be high, because the set we are talking about, although finite, tends to be *big*, i.e., to have exponential size.

The Exhaustive Search Paradigm

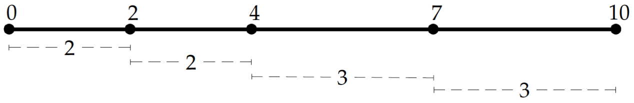
- Exhaustive Search algorithms, also called *brute force algorithms*, are a sort of algorithms which:
  - Which typically have *high* (most often, exponential) complexity .
  - But which are often relatively easy to be proved correct.
- The idea behind an exhaustive search algorithm is that, whenever the problem *can be seen* as the problem of looking for an element in a finite set:
  - **Having** certain properties;
  - or being **the best** according to a given notion of optimality;
  - or cominations thereof.
- The complexity tend to be high, because the set we are talking about, although finite, tends to be *big*, i.e., to have exponential size.

Restriction Mapping

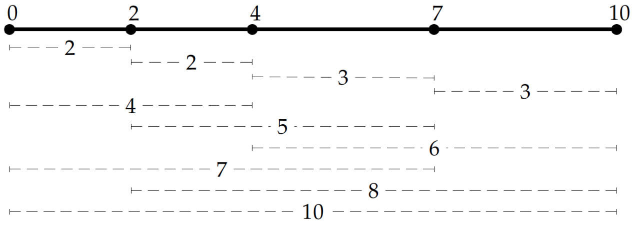
- Suppose you know the distances between all the exits along a turnpike, and you want to reconstruct the *map* of the turnpike.
  - A similar problem occurs in genomics, where the turnpike is a DNA strand, and the exits are the occurrence of a specific sequence.
- A *multiset* is like a set, but allows for duplicate elements.
  - The multisets $\{2, 2, 3, 4\}$ and $\{2, 3, 4, 4\}$ are different. When seen as sets, they are instead the same.
- Given a set of points X, $\Delta(X)$ stands for the multiset of distances between the points in X.

Restriction Mapping

- Suppose you know the distances between all the exits along a turnpike, and you want to reconstruct the *map* of the turnpike.
  - A similar problem occurs in genomics, where the turnpike is a DNA strand, and the exits are the occurrence of a specific sequence.
- A *multiset* is like a set, but allows for duplicate elements.
  - The multisets $\{2, 2, 3, 4\}$ and $\{2, 3, 4, 4\}$ are different. When seen as sets, they are instead the same.
- Given a set of points X, $\Delta(X)$ stands for the multiset of distances between the points in X.



(a) Complete digest.



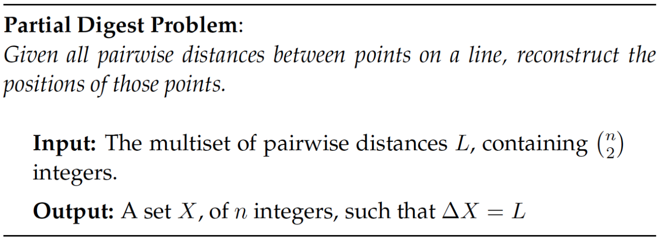
(b) Partial digest.

The Partial Digest Problem

- The **Partial Digest Problem** consists in reconstruct X from ΔX, keeping in mind that
  - If X has n elements, ΔX has
  $$\binom{n}{2} = \frac{n(n-1)}{2}$$
  points
  - There could be $X \neq Y$ such that $\Delta X = \Delta Y$.

The Partial Digest Problem

- The **Partial Digest Problem** consists in reconstruct X from ΔX, keeping in mind that
  - If X has n elements, ΔX has
    $$\binom{n}{2} = \frac{n(n-1)}{2}$$
    points
  - There could be $X \neq Y$ such that $\Delta X = \Delta Y$.



Partial Digest Problem:
Given all pairwise distances between points on a line, reconstruct the positions of those points.

Input: The multiset of pairwise distances L, containing $\binom{n}{2}$ integers.

Output: A set X, of n integers, such that $\Delta X = L$

The Trivial Brute Force Algorithm

```
BruteForcePDP($L, n$)
1  $M \leftarrow$ maximum element in $L$
2  for every set of $n - 2$ integers $0 < x_2 < \cdots < x_{n-1} < M$
3      $X \leftarrow \{0, x_2, \ldots, x_{n-1}, M\}$
4      Form $\Delta X$ from $X$
5      if $\Delta X = L$
6          return $X$
7  output "No Solution"
```

Correctness and Complexity

- The **correctness** of the brute force algorithms can be proved easily: of course among the (many) sequences considered, there is *the one* generating L.
  - There could be many, but of course we have

    $$\Delta X = \Delta(X \oplus v)$$

    where $X \oplus v = \{x + v \mid x \in X\}$. As a consequence, it is safe to take one of the points in X to be 0.
- About its **complexity**, the number of iterations of the algorithm is the number of distinct ways one can pick $n - 2$ elements from a set of $M - 1$ elements is

$$\binom{M-1}{n-2} = O(M^{n-2}).$$

A Better Brute Force Algorithm

- One may wonder why the numbers $x_2, \ldots, x_{n-1}$ are chosen to be arbitrary numbers.
- Indeed, we can restrict them to be (distinct) elements of L, because one of the extremes is chosen to be 0.
- The obtained algorithm examines

$$\binom{|L|}{n-2} = O(n^{2n-4})$$

  different sets of integers, since $|L| = \frac{n(n-1)}{2}$.

A Better Brute Force Algorithm

- One may wonder why the numbers $x_2, \dots, x_{n-1}$ are chosen to be arbitrary numbers.
- Indeed, we can restrict them to be (distinct) elements of L, because one of the extremes is chosen to be 0.
- The obtained algorithm examines

$$\binom{|L|}{n-2} = O(n^{2n-4})$$

different sets of integers, since $|L| = \frac{n(n-1)}{2}$.

```
ANOTHERBRUTEFORCEPDP($L, n$)
1  $M \leftarrow$ maximum element in $L$
2  for every set of $n-2$ integers $0 < x_2 < \cdots < x_{n-1} < M$ from $L$
3      $X \leftarrow \{0, x_2, \dots, x_{n-1}, M\}$
4      Form $\Delta X$ from $X$
5      if $\Delta X = L$
6          return $X$
7  output "No Solution"
```

An Incremental Strategy

$L = \{2, 3, 5, 7, 8, 10\}$	$X = \{0\}$
$L = \{2, 3, 5, 7, 8\}$	$X = \{0, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
Impossible!	$X = \{0, 2, 3, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
$\emptyset$	$X = \{0, 2, 7, 10\}$

An Incremental Strategy

$L = \{2, 3, 5, 7, 8, 10\}$	$X = \{0\}$
$L = \{2, 3, 5, 7, 8\}$	$X = \{0, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
Impossible!	$X = \{0, 2, 3, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
$\emptyset$	$X = \{0, 2, 7, 10\}$

An Incremental Strategy

$L = \{2, 3, 5, 7, 8, 10\}$	$X = \{0\}$
$L = \{2, 3, 5, 7, 8\}$	$X = \{0, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
Impossible!	$X = \{0, 2, 3, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
$\emptyset$	$X = \{0, 2, 7, 10\}$

An Incremental Strategy

$L = \{2, 3, 5, 7, 8, 10\}$	$X = \{0\}$
$L = \{2, 3, 5, 7, 8\}$	$X = \{0, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
Impossible!	$X = \{0, 2, 3, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
$\emptyset$	$X = \{0, 2, 7, 10\}$

An Incremental Strategy

$L = \{2, 3, 5, 7, 8, 10\}$	$X = \{0\}$
$L = \{2, 3, 5, 7, 8\}$	$X = \{0, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
Impossible!	$X = \{0, 2, 3, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
$\emptyset$	$X = \{0, 2, 7, 10\}$

An Incremental Strategy

$L = \{2, 3, 5, 7, 8, 10\}$	$X = \{0\}$
$L = \{2, 3, 5, 7, 8\}$	$X = \{0, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
Impossible!	$X = \{0, 2, 3, 10\}$
$L = \{3, 5, 7\}$	$X = \{0, 2, 10\}$
$\emptyset$	$X = \{0, 2, 7, 10\}$

A Practical Algorithm

```
PartialDigest($L$)
1  $width \leftarrow$ Maximum element in $L$
2  Delete($width, L$)
3  $X \leftarrow \{0, width\}$
4  Place($L, X$)

Place($L, X$)
 1  if $L$ is empty
 2      output $X$
 3      return
 4  $y \leftarrow$ Maximum element in $L$
 5  if $\Delta(y, X) \subseteq L$
 6      Add $y$ to $X$ and remove lengths $\Delta(y, X)$ from $L$
 7      Place($L, X$)
 8      Remove $y$ from $X$ and add lengths $\Delta(y, X)$ to $L$
 9  if $\Delta(width - y, X) \subseteq L$
10      Add $width - y$ to $X$ and remove lengths $\Delta(width - y, X)$ from $L$
11      Place($L, X$)
12      Remove $width - y$ from $X$ and add lengths $\Delta(width - y, X)$ to $L$
13  return
```

Correctness and Complexity

- Place is the typical example of a so-called **backtracking** algorithm: when we realize that some of the choices we have previously done are wrong, we need to backtrack.
- The proof of **correctness** of this algorithm goes by induction on $|L|$, but in order to prove it, we need to strengthen the induction hypothesis, as usual.
- About its **complexity**, we can only say it remains exponential *in the worst case*. The following recurrence relation expresses the worst-case number of instructions:

  $$T(n) \leq 2T(n-1) + cn$$

  whose solution is an exponential (as in the case of Hanoi's towers.

Regulatory Motifs in DNA Sequences

- Suppose you have a long DNA sequence s, and you know that *some* substring of length l occurs many times in the string, perhaps slightly altered.
- There are many problems one could be interested at, and in particular:
  1. Finding *where* the occurrences of the substring are located.
  2. Determining *the substring* itself.
- For the sake of simplicity, we assume that each occurrence of the substring in s is in a difference region of s.
  - As a consequence, we will work on *sequences* of strings, rather than with strings.

Random Sequences

```
CGGGGCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAACCAAAGCGGACAAA
GGGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCTC
CTGCTGTACAACTGAGATCATGCTGCTTCAAC
TACATGATCTTTTGTGGATGAGGGAATGATGC
```

Implanting One Substring

CGGGGCT$\underline{\texttt{ATGCAACT}}$GGGTCGTCACATTCCCCTTTCGATA

TTTGAGGGTGCCCAATAA$\underline{\texttt{ATGCAACT}}$CCAAAGCGGACAAA

GG$\underline{\texttt{ATGCAACT}}$GATGCCGTTTGACGACCTAAATCAACGGCC

AAGG$\underline{\texttt{ATGCAACT}}$CCAGGAGCGCCTTTGCTGGTTCTACCTG

AATTTTCTAAAAAGATTATAATGTCGGTCC$\underline{\texttt{ATGCAACT}}$TC

CTGCTGTACAACTGAGATCATGCTGC$\underline{\texttt{ATGCAACT}}$TTCAAC

TACATGATCTTTTG$\underline{\texttt{ATGCAACT}}$TGGATGAGGGAATGATGC

Implanting One Substring

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

Implanting Approximate Substrings

```
CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
GGATGgAtCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAaGCAACcCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC
```

How to even *Formulate* the Problem?

- Rather than directly looking for "approximate" occurrences of a substring, we can define the *score* and *consensus string* of any sequence of positions.
- Formally, given a $t \times n$ matrix, called DNA, and a natural number $l \leq n$, we can define:
  - A *sequence of starting positions* as a sequence $s = (s_1, s_2, \ldots, s_t)$ such that $1 \leq s_i \leq n - l$.
  - The *profile* matrix $\mathbf{P}(s)$ as the $4 \times l$ matrix of natural numbers whose elements count the number of occurrences of each DNA character in the matrix, starting at s. $M_{\mathbf{P}(s)}(j)$ is the largest count in column j in $\mathbf{P}(s)$.
  - The *consensus* string for s is the most likely string of length l, given s.
  - The *score* of s is just $\sum_{j=1}^{l} M_{\mathbf{P}(s)}(j)$, and is indicated as $Score(s, DNA)$.

How to even *Formulate* the Problem?

- Rather than directly looking for "approximate" occurrences of a substring, we can define the *score* and *consensus string* of any sequence of positions.
- Formally, given a $t \times n$ matrix, called DNA, and a natural number $l \leq n$, we can define:
  - A *sequence of starting positions* as a sequence $s = (s_1, s_2, \ldots, s_t)$ such that $1 \leq s_i \leq n - l$.
  - The *profile* matrix $\mathbf{P}(s)$ as the $4 \times l$ matrix of natural numbers whose elements count the number of occurrences of each DNA character in the matrix, starting at s. $M_{\mathbf{P}(s)}(j)$ is the largest count in column j in $\mathbf{P}(s)$.
  - The *consensus* string for s is the most likely string of length l, given s.
  - The *score* of s is just $\sum_{j=1}^{l} M_{\mathbf{P}(s)}(j)$, and is indicated as $Score(s, DNA)$.

String's Superposition

```
                       CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTT...
            TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
                            GGATGgAtCTGATGCCGTTTGACGACCTA...
                          AAGGAaGCAACcCCAGGAGCGCCTTTGCTGG...
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
    CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
                TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC
```

The Alignment's Matrix

Alignment		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
Profile	**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
Consensus		A	T	G	C	A	A	C	T

Motif Finding Problem:
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 $\mathbf{s} = (s_1, s_2, \ldots, s_t)$ maximizing $Score(\mathbf{s}, DNA)$.

Median Strings

- A concept which is very much related to that of motifs and consensus strings is that of a median strings.
- Given two strings of the same length u and v, their *Hamming distance* $d_H(u, v)$ is the number of positions at which they differ.
  - This can be generalised to the distance $d_H(u, s)$ between a string u and a sequence of positions $s = (s_1, \ldots, s_t)$.
- The **total distance** between a string u and a $t \times n$ matrix DNA is defined as

$$TotalDistance(u, DNA) = \min_{s} d_H(u, s)$$

- When looking for a string (approximately) occurring in DNA, one can simply look for a u minimizing $TotalDistance(u, DNA)$.

Median String Problem:
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.

Motif Finding vs. Median Strings

$$d_H(w, \mathbf{s}) = lt - Score(\mathbf{s}, DNA).$$

Motif Finding vs. Median Strings

$$d_H(w, \mathbf{s}) = lt - Score(\mathbf{s}, DNA).$$

$$d_H(w, \mathbf{s}) = \min_{\text{all choices of } v} d_H(v, \mathbf{s}) = lt - Score(\mathbf{s}, DNA)$$

Motif Finding vs. Median Strings

$$d_H(w, \mathbf{s}) = lt - Score(\mathbf{s}, DNA).$$

$$d_H(w, \mathbf{s}) = \min_{\text{all choices of } v} d_H(v, \mathbf{s}) = lt - Score(\mathbf{s}, DNA)$$

$$\min_{\text{all choices of } \mathbf{s}} \min_{\text{all choices of } v} d_H(v, \mathbf{s}) = lt - \max_{\text{all choices of } \mathbf{s}} Score(\mathbf{s}, DNA).$$

Motif Finding vs. Median Strings

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**

Two Brute Force Algorithms

- In the **Motif Finding** problem, we could proceed by considering all possible positions s, and computing its score.
  - The number of those strings is $(n - l + 1)^t$.
- In the **Median String** problem, we could instead proceed by considering all possible 4^l possible strings, computing for each of it its total total distance to DNA.
  - The number of those strings is 4^l.
- Can we do better than that? Can we perform *significantly less* operations than the one given by the bounds above?
  - It is not clear how one can achieve that: we need to consider all positions, and all strings, respectively.

Two Brute Force Algorithms

- In the **Motif Finding** problem, we could proceed by considering all possible positions s, and computing its score.
  - The number of those strings is $(n - l + 1)^t$.
- In the **Median String** problem, we could instead proceed by considering all possible 4^l possible strings, computing for each of it its total total distance to DNA.
  - The number of those strings is 4^l.
- Can we do better than that? Can we perform *significantly less* operations than the one given by the bounds above?
  - It is not clear how one can achieve that: we need to consider all positions, and all strings, respectively.

Two Brute Force Algorithms

- In the **Motif Finding** problem, we could proceed by considering all possible positions s, and computing its score.
  - The number of those strings is $(n - l + 1)^t$.
- In the **Median String** problem, we could instead proceed by considering all possible 4^l possible strings, computing for each of it its total total distance to DNA.
  - The number of those strings is 4^l.
- Can we do better than that? Can we perform *significantly less* operations than the one given by the bounds above?
  - It is not clear how one can achieve that: we need to consider all positions, and all strings, respectively.

A Brute-Force Algorithm for the Motif Finding Problem

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

A Brute-Force Algorithm for the Median String Problem

```
BruteForceMedianSearch($DNA, t, n, l$)
1   $bestWord \leftarrow$ AAA $\cdots$ AA
2   $bestDistance \leftarrow \infty$
3   for each $l$-mer $word$ from AAA...A to TTT...T
4       if TotalDistance($word, DNA$) $< bestDistance$
5           $bestDistance \leftarrow$ TotalDistance($word, DNA$)
6           $bestWord \leftarrow word$
7   return $bestWord$
```

Strings as Tuples

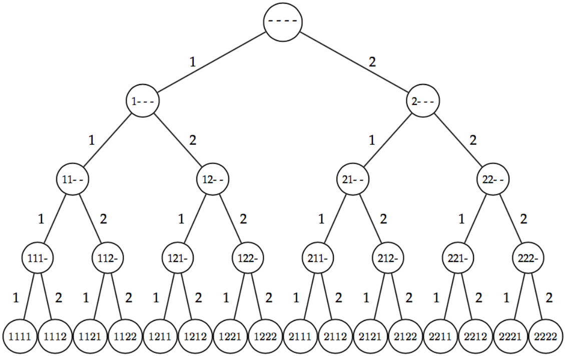
AA$\cdots$ AA	$(1, 1, \ldots, 1, 1)$
AA$\cdots$ AT	$(1, 1, \ldots, 1, 2)$
AA$\cdots$ AG	$(1, 1, \ldots, 1, 3)$
AA$\cdots$ AC	$(1, 1, \ldots, 1, 4)$
AA$\cdots$ TA	$(1, 1, \ldots, 2, 1)$
AA$\cdots$ TT	$(1, 1, \ldots, 2, 2)$
AA$\cdots$ TG	$(1, 1, \ldots, 2, 3)$
AA$\cdots$ TC	$(1, 1, \ldots, 2, 4)$
$\vdots$	$\vdots$
CC$\cdots$ GG	$(4, 4, \ldots, 3, 3)$
CC$\cdots$ GC	$(4, 4, \ldots, 3, 4)$
CC$\cdots$ CA	$(4, 4, \ldots, 4, 1)$
CC$\cdots$ CT	$(4, 4, \ldots, 4, 2)$
CC$\cdots$ CG	$(4, 4, \ldots, 4, 3)$
CC$\cdots$ CC	$(4, 4, \ldots, 4, 4)$

Trees

- The way we see strings as tuples enables us to see the explore the space of all strings of a given length as the leaves of a tree.

Trees

- The way we see strings as tuples enables us to see the explore the space of all strings of a given length as the leaves of a tree.



Trees

- Trees are pervasive in computer science, and the *branching* analogue of sequences, which are instead linear.
- The kind of trees we are interested at here are such that all leaves have the same height h, and all nodes have either *a fixed number k of children* or *no children* at all.
- The total number of leaves is precisely h^k in this case.

Traveling Inside a Tree

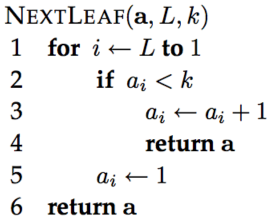
- Visiting all the leaves in a tree is thus a way to enumerate all the strings of a certain length in a given alphabet.
- How could we jump from a given leaf to “the next one”?

Traveling Inside a Tree

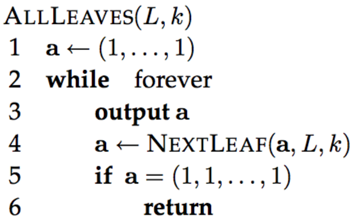
- Visiting all the leaves in a tree is thus a way to enumerate all the strings of a certain length in a given alphabet.
- How could we jump from a given leaf to “the next one”?

Traveling Inside a Tree

- Visiting all the leaves in a tree is thus a way to enumerate all the strings of a certain length in a given alphabet.
- How could we jump from a given leaf to “the next one”?



```
NEXTLEAF($\mathbf{a}, L, k$)
for $i \leftarrow L$ to $1$
    if $a_i < k$
        $a_i \leftarrow a_i + 1$
        return $\mathbf{a}$
    $a_i \leftarrow 1$
return $\mathbf{a}$
```



```
ALLLEAVES($L, k$)
$\mathbf{a} \leftarrow (1, \ldots, 1)$
while forever
    output $\mathbf{a}$
    $\mathbf{a} \leftarrow$ NEXTLEAF($\mathbf{a}, L, k$)
    if $\mathbf{a} = (1, 1, \ldots, 1)$
        return
```

A New Way of Formulating the Brute Force Algorithm

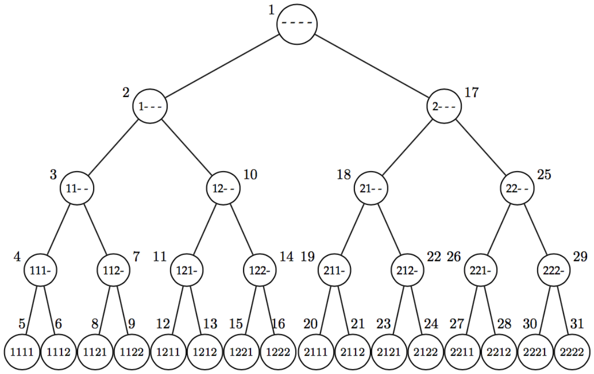
```
BruteForceMotifSearchAgain($DNA, t, n, l$)
1  $\mathbf{s} \leftarrow (1, 1, \ldots, 1)$
2  $bestScore \leftarrow Score(\mathbf{s}, DNA)$
3  while forever
4        $\mathbf{s} \leftarrow$ NextLeaf($\mathbf{s}, t, n - l + 1$)
5        if $Score(\mathbf{s}, DNA) > bestScore$
6              $bestScore \leftarrow Score(\mathbf{s}, DNA)$
7              bestMotif $\leftarrow (s_1, s_2, \ldots, s_t)$
8        if $s = (1, 1, \ldots, 1)$
9              return bestMotif
```

Visiting the Whole Tree

- Now, suppose we want to visit the whole tree, rather than just its leaves.
- We would like to first visit a node, then the sub-tree rooted at its left, and then the sub-tree rooted at its right.

Visiting the Whole Tree

- Now, suppose we want to visit the whole tree, rather than just its leaves.
- We would like to first visit a node, then the sub-tree rooted at its left, and then the sub-tree rooted at its right.



Visiting the Whole Tree

- How could we find the next *vertex* in the tree (as opposed to the next *leaf* in the tree?
- If you are at a level i, there are cases in which you want to go *down*, and cases in which you need to go *up*.

Visiting the Whole Tree

- How could we find the next *vertex* in the tree (as opposed to the next *leaf* in the tree?
- If you are at a level i, there are cases in which you want to go *down*, and cases in which you need to go *up*.

```
NextVertex($\mathbf{a}, i, L, k$)
if $i < L$
    $a_{i+1} \leftarrow 1$
    return $(\mathbf{a}, i+1)$
else
    for $j \leftarrow L$ to 1
        if $a_j < k$
            $a_j \leftarrow a_j + 1$
            return $(\mathbf{a}, j)$
return $(\mathbf{a}, 0)$
```

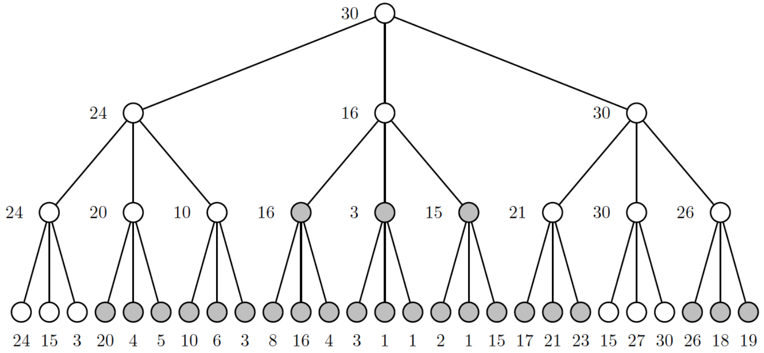
- i is the level of the tree in which you currently are;
- L is the height of the tree;
- k is the size of the underlying set.

Visiting The Whole Tree

- If you just replace NextLeaf by NextVertex, one gets an algorithm which is not particularly clever, because it also visits the internal nodes.

```
SimpleMotifSearch($DNA, t, n, l$)
1  $\mathbf{s} \leftarrow (1, \ldots, 1)$
2  $bestScore \leftarrow 0$
3  $i \leftarrow 1$
4  while $i > 0$
5      if $i < t$
6          $(\mathbf{s}, i) \leftarrow$ NextVertex($\mathbf{s}, i, t, n - l + 1$)
7      else
8          if $Score(\mathbf{s}, DNA) > bestScore$
9              $bestScore \leftarrow Score(\mathbf{s}, DNA)$
10             bestMotif $\leftarrow (s_1, s_2, \ldots, s_t)$
11         $(\mathbf{s}, i) \leftarrow$ NextVertex($\mathbf{s}, i, t, n - l + 1$)
12 return bestMotif
```

An Interesting Tree



Avoiding Useless Work

```
BranchAndBoundMotifSearch($DNA, t, n, l$)
$\mathbf{s} \leftarrow (1, \ldots, 1)$
$bestScore \leftarrow 0$
$i \leftarrow 1$
while $i > 0$
    if $i < t$
        $optimisticScore \leftarrow Score(\mathbf{s}, i, DNA) + (t - i) \cdot l$
        if $optimisticScore < bestScore$
            $(\mathbf{s}, i) \leftarrow$ Bypass($\mathbf{s}, i, t, n - l + 1$)
        else
            $(\mathbf{s}, i) \leftarrow$ NextVertex($\mathbf{s}, i, t, n - l + 1$)
    else
        if $Score(\mathbf{s}, DNA) > bestScore$
            $bestScore \leftarrow Score(\mathbf{s})$
            bestMotif $\leftarrow (s_1, s_2, \ldots, s_t)$
        $(\mathbf{s}, i) \leftarrow$ NextVertex($\mathbf{s}, i, t, n - l + 1$)
return bestMotif
```

- With Score($\mathbf{s}, i, DNA$), we compute the score *of the first* i positions in i;
- The score of *the other ones* can be at most $(t - i) \cdot l$.
- As a consequence, if *optimisticScore* is strictly less than *bestScore*, we can bypass the tree rooted at the current node.

- Branch and Bound techniques can be quite effective, although the worst-case complexity stays exponential.

Thank You!

Questions?