Outline

- Signature Files
  - Signature for attribute values
  - Signature for records
  - Searching a signature file
- Signature Trees
  - Signature tree construction
  - Searching a signature tree
  - About balanced signature trees

- **Signature file**
  - A signature file is a set of bit strings, which are called *signatures*.
  - In a signature file, each signature is constructed for a record in a table, a block of text, or an image.
  - When a query arrives, a query signature will be constructed according to the key words involved in the query. Then, the signature file will be searched against the query signature to discard non-qualifying signatures, as well as the objects represented by those signatures.

• **Signature file**

- Generate a signature for an attribute value

Before we generate the signature for an attribute value, three parameters have to be determined

F: number of 1s in bit string
m: length of bit string
D: number of attribute values in a record (or average number of the key words of in a block of text)

Optimal choice of the parameters:

$$m \times \ln 2 = F \times D$$

•Signature file

- Decompose an attribute value (or a key word) into a series of triplets
- Using a hash function to map a triplet to an integer *p*, indicating that the *p*th bit in the signature will be set to 1.

Example: Consider the word “professor”. We will decompose it into 6 triplets:
“pro”, “rof”, “ofe”, “fes”, “ess”, “sor”.
Assume that hash(pro) = 2, hash(rof) = 4, hash(ofe) =8, and hash(fes) = 9.

Signature: 010 100 011 000

- **Signature file**
  - Generate a signature for a record (or a block of text)

block: ... SGML ... databases ... information ...

word signature:

SGML		010 000 100 110
database		100 010 010 100
information	$\vee$	010 100 011 000
object signature (OS)		110 110 111 110

superimposing

- **Signature file**
  - Generate a signature for a record (or a block of text)

relation:

name	sex	
John	male	… …
…	…	

signature file:

s_1	1011 0110
s_2	1011 1001
s_3	1010 0111
s_4	0111 0110
s_5	0111 0101
s_6	0101 1100
s_7	1110 0100
s_8	1010 1011

- **Signature file**
  - Search a signature file

    When a query arrives, the query signature will be constructed and the object signatures are scanned and many non-qualifying objects are discarded.
  - When comparing the query signature s_q and an object signature s, three possible outcomes: (1) the object matches the query; that is, for every bit set in s_q, the corresponding bit in the object signature s is also set (i.e., $s \wedge s_q = s_q$) and the object contains really the query word; (2) the object doesn't match the query (i.e., $s \wedge s_q \neq s_q$); and (3) the signature comparison indicates a match but the object in fact doesn't match the search criteria (false drop).

- **Signature file**
  - Search a signature file

block: ... SGML ... databases ... information ...

object signature (OS): 110 110 111 110

queries:	query signatures:	matching results:
SGML	010 000 100 110	match with OS
XML	011 000 100 100	no match with OS
informatik	110 100 100 000	false drop

- **Signature file**
  - Search a signature file

s_1	1011 0110
s_2	1011 1001
s_3	1010 0111
s_4	0111 0110
s_5	0111 0101
s_6	0101 1100
s_7	1110 0100
s_8	1010 1011

query: John $\wedge$ male

$\Downarrow$

query signature: 1010 0101

- **Signature tree**
  - Signature tree construction

Consider a signature s_i of length m. We denote it as $s_i = s_i[1].. s_i[m]$, where each $s_i[j] \in \{0, 1\}$ ($j = 1, ..., m$). We also use $s_i(j_1, ..., j_h)$ to denote a sequence of pairs with respect to s_i: $(j_1, s_i[j_1])(j_2, s_i[j_2])$ ... $(j_h, s_i[j_h])$, where $1 \leq j_k \leq m$ for $k \in \{1, ..., h\}$.

Definition (*signature identifier*) Let $S = s_1.s_2 ... .s_n$ denote a signature file. Consider s_i ($1 \leq i \leq n$). If there exists a sequence: $j_1, ..., j_h$ such that for any $k \neq i$ ($1 \leq k \leq n$) we have $\boldsymbol{s_i(j_1, ..., j_h)} \neq s_k(j_1, ..., j_h)$, then we say $s_i(j_1, ..., j_h)$ identifies the signature s_i or say $s_i(j_1, ..., j_h)$ is an identifier of s_i.

- **Signature tree**
  - Signature tree construction

s_1	1011 0110
s_2	1011 1001
s_3	1010 0111
s_4	0111 0110
s_5	0111 0101
s_6	0101 1100
s_7	1110 0100
s_8	1010 1011

Example:

$s_8(5, 1, 4) = (5, 1)(1, 1)(4, 0)$

For any $i \neq 8$ **we have** $s_i(5, 1, 4) \neq s_8(5, 1, 4)$**. For instance,** $s_5(5, 1, 4) = (5, 0)(1, 0)(4, 1) \neq s_8(5, 1, 4)$, $s_2(5, 1, 4) = (5, 1)(1, 1)(4, 1) \neq s_8(5, 1, 4)$**, and so on.**

$s_1(5, 4, 1) = (5, 0)(4, 1)(1, 1)$

For any $i \neq 1$ **we have** $s_i(5, 4, 1) \neq s_1(5, 4, 1)$**.**

- **Signature tree**
  - Signature tree construction

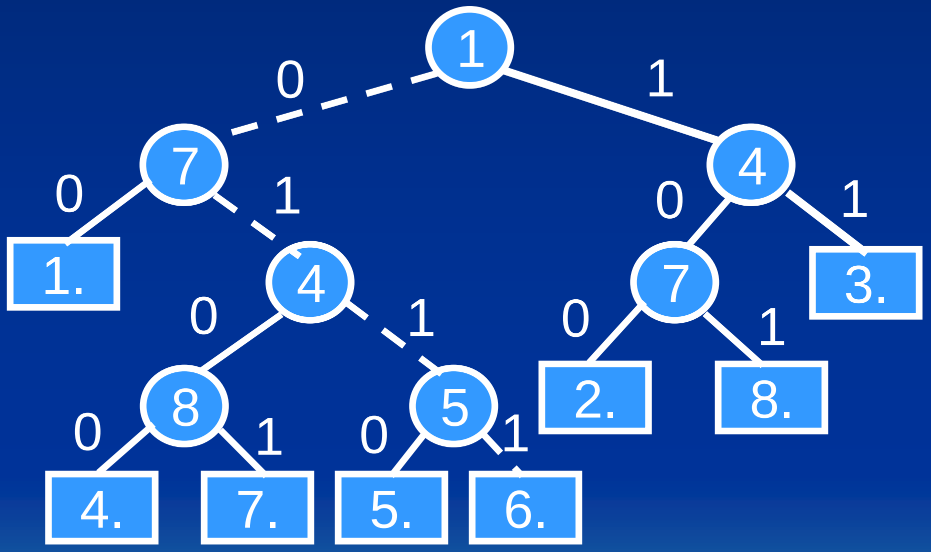
Definition (*signature tree*) A signature tree for a signature file $S = s_1.s_2 \ldots .s_n$, where $s_i \neq s_j$ for $i \neq j$ and $|s_k| = m$ for $k = 1, \ldots, n$, is a binary tree T such that

1. For each internal node of T, the left edge leaving it is always labeled with 0 and the right edge is always labeled with 1.
2. T has n leaves labeled 1, 2, ..., n, used as pointers to n different positions of s_1, s_2, ... and s_n in S. Let v be a leaf node. Denote $p(v)$ the pointer to the corresponding signature.
3. Each internal node v is associated with a number, denoted $sk(v)$, to tells which bit will be checked.
4. Let $i_1, \ldots, i_h$ be the numbers associated with the nodes on a path from the root to a leaf v labeled i (then, this leaf node is a pointer to the ith signature in S, *i.e.*, $p(v) = i$). Let $p_1, \ldots, p_h$ be the sequence of labels of edges on this path. Then, $(j_1, p_1) \ldots (j_h, p_h)$ makes up a signature identifier for s_i, $s_i(j_1, \ldots, j_h)$.

- **Signature tree**

  - Signature tree construction

s_1	011 001 000 101
s_2	111 011 001 111
s_3	111 101 010 111
s_4	011 001 101 111
s_5	011 101 110 101
s_6	011 111 110 101
s_7	011 001 111 111
s_8	111 011 111 111



```
Algorithm sig-tree-generation(file)
begin
    construct a root node r with sk(r) = 1;
            /*where r corresponds to the first signature s1 in the signature file*/
    for j = 2 to n do
     call insert(sj);
end
Procedure insert(s)
begin
    stack ← root;
    while stack not empty do
      {v ← pop(stack);
        if v is not a leaf then
            {i ← sk(v);
             if s[i] = 1 then
            {let a be the right child of v; push(stack, a);}
             else {let a be the left child of v; push(stack, a);}
            }
        else                                    (*v is a leaf.*)
```

{ compare s with the signature s_0 pointed to by $p(v)$;

assume that the first k bit of s agree with s_0;

but s differs from s_0 in the $(k + 1)$th position;

$w \leftarrow v$; replace v with a new node u with $sk(u) = k + 1$;

if $s[k + 1] = 1$ **then**

make s and w be respectively the right and left children of u

else make w and s be the right and left children of u, respectively;}

}

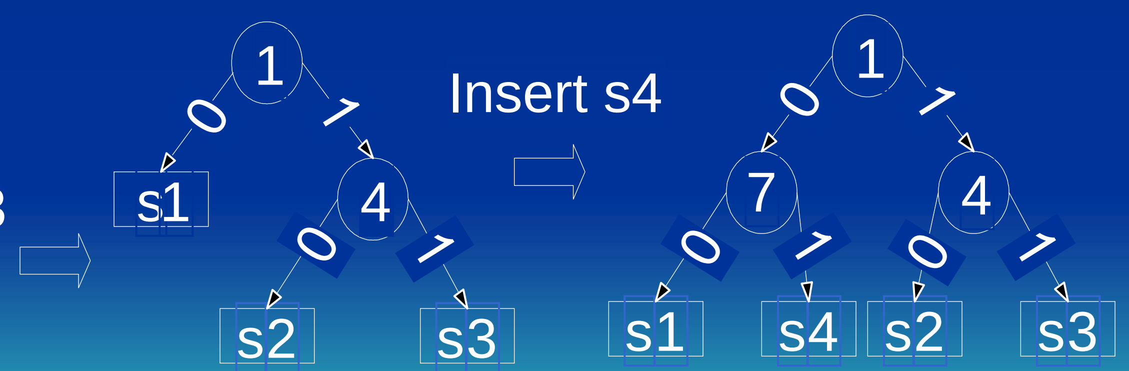
end

- **Signature tree**

  - Signature tree construction

S1	011001000101
S2	111011001111
S3	111101010111
S4	011001101111

Signature file



- **Signature tree**
  - Searching of a signature tree

Let s_q be a query signature. The *i*th position of s_q is denoted as $s_q[i]$. During the traversal of a signature tree, the inexact matching is done as follows:

(i) Let v be the node encountered and $s_q[i]$ be the position to be checked.

(ii) If $s_q[i] = 1$, we move to the right child of v.

(iii) If $s_q[i] = 0$, both the right and left child of v will be explored.

Algorithm *signature-tree-search*

input: a query signature *sq*;

output: a set of signatures which survive the checking;

1. $R \leftarrow \varnothing$.
2. Push the root of the signature tree into $stack_p$.
3. If $stack_p$ is not empty, $v \leftarrow$ pop($stack_p$); else return(R).
4. If v is not a leaf node, $i \leftarrow sk(v)$;

   If s_q (i) = 0, push c_r and c_l into $stack_p$;

   (where c_r and c_l are v's right and left child, respectively.)

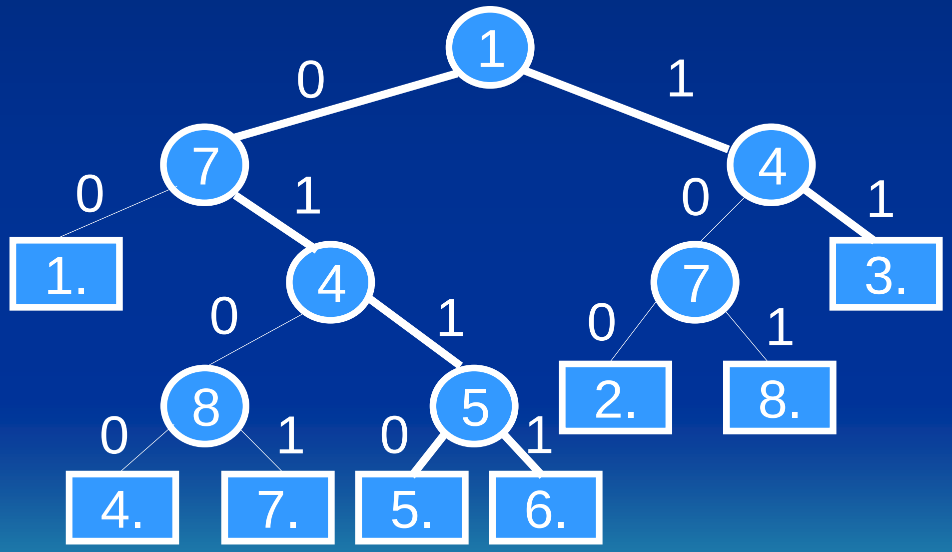
   otherwise, push only c_r into $stack_p$.
5. Compare s_q with the signature pointed by $p(v)$.

   /*$p(v)$ - pointer to the block signature*/

   If s_q matches, $R \leftarrow R \cup \{p(v)\}$.
6. Go to (3).

- **Signature tree**
  - Searching of a signature tree

    query signature: s_q = 000 100 100 000.



- **Signature tree**
  - About balanced signature trees

    A signature tree can be quite skewed.

S1: 100 100 100 100
S2: 010 010 010 010
S3: 001 001 001 001
S4: 000 110 010 010
S5: 000 011 001 001
S6: 000 001 100 100
S7: 000 000 110 010
S8: 000 000 010 110

- **Signature tree**
  - About balanced signature trees

    Weight-based method:

    A signature file $S = s_1.s_2 \ldots .s_n$ can be considered as a boolean matrix. We use $S[i]$ to represent the ith column of S. We calculate the weight of each $S[i]$, *i.e.*, the number of 1s appearing in $S[i]$, denoted $w(S[i])$. Then, we choose an j such that $|w(S[i]) - n/2|$ is minimum. Here, the tie is resolved arbitrarily. Using this j, we divide S into two groups $g_1 = \{ s_{i_1}$, ..., s_{i_k} } with each $s_{i_p}[j] = 0$ ($p = 1, \ldots, k$) and $g_2 = \{ s_{i_{k+1}}$, ..., s_{i_n} } with each $s_{i_q}[j] = 1$ ($q = k + 1, \ldots, n$).

- **Signature tree**
  - About balanced signature trees

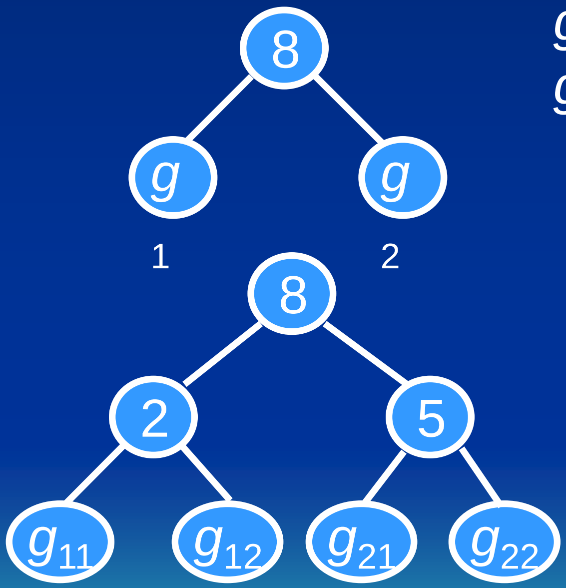
    Weight-based method (continued):

In a next step, we consider each g_i (i = 1, 2) as a single signature file and perform the same operations as above, leading to two trees generated for g_1 and g_2, respectively. Replacing g_1 and g_2 with the corresponding trees, we get another tree. We repeat this process until the leaf nodes of a generated tree cannot be divided any more.

- **Signature tree**
  - About balanced signature trees

    Example:

S1: 100 100 100 100
S2: 010 010 010 010
S3: 001 001 001 001
S4: 000 110 010 010
S5: 000 011 001 001
S6: 000 001 100 100
S7: 000 000 110 010
S8: 000 000 010 110



$g_1 = \{s_1, s_3, s_5, s_6\}$
$g_2 = \{s_2, s_4, s_7, s_8\}$

$g_{11} = \{s_3, s_5\}$
$g_{12} = \{s_6, s_1\}$
$g_{21} = \{s_8, s_7\}$
$g_{22} = \{s_4, s_2\}$

- **Signature tree**
  - About balanced signature trees

Algorithm *balanced-tree-generation*(*file*)
input: a signature file.
output: a signature tree.
Begin
let $S = file$; $N \leftarrow |S|$;
if $N > 1$ **then** {
choose j such that $|w(S[j]) - N/2|$ is minimum;
let $g_1 = \{s_{i_1}, \ldots, s_{i_k}\}$ with each $s_{i_p}[j] = 0$ ($p = 1, \ldots, k$);
let $g_2 = \{s_{i_{k+1}}, , \ldots, s_{i_{k+N}}\}$ with each $s_{i_q}[j] = 1$ ($q = k + 1, \ldots, N$)

generate a tree containing a root *r* and two child nodes marked with

g_1 and g_2, respectively;

skip(*r*) $\leftarrow$ *j*;

replace the node marked *g*1 with *balanced-tree-generation*(g_1);

replace the node marked *g*2 with *balanced-tree-generation*(g_2);}

else return;

end