CMPSCI 546 (590R)
Applied Information Retrieval

Indexing

V-Byte Encoder

```
public void encode ( int [] input, ByteBuffer output) {
   for (int i : input ) {
       while ( i >= 128 ) {
             output.put ( i & 0x7F ) ;
             i >>>= 7 ; // logical shift, no sign bit extension
        }
        output.put( i | 0x80 );
   }
}
```

V-Byte Decoder

```
public void decode ( byte [] input, IntBuffer output ) {
  for ( int i = 0; i < input.length; i++ ) {
     int position = 0;
     int result = ((int) input[i] & 0x7F);

     while ( (input[i]] & 0x80) == 0 ) {
         i += 1;
         position += 1;
         int unsignedByte = ((int) input[i] & 0x7F);
         result |= (unsignedByte << (7 * position));
     }
     output.put(result);
  }
}
```

Auxiliary Structures

- Inverted lists usually stored together in a single file for efficiency
  - *Inverted file*
- *Vocabulary* or *lexicon*
  - Contains a lookup table from index terms to the byte offset of the inverted list in the inverted file
  - Either hash table in memory or B-tree for larger vocabularies
- Term statistics stored at start of inverted lists
- Collection statistics stored in separate file

Index Construction

- Simple in-memory indexer

```
procedure BuildIndex($D$)                                  ▷ $D$ is a set of text documents
    $I \leftarrow$ HashTable()                               ▷ Inverted list storage
    $n \leftarrow 0$                                         ▷ Document numbering
    for all documents $d \in D$ do
        $n \leftarrow n + 1$
        $T \leftarrow$ Parse($d$)                            ▷ Parse document into tokens
        Remove duplicates from $T$
        for all tokens $t \in T$ do
            if $I_t \notin I$ then
                $I_t \leftarrow$ Array()
            end if
            $I_t$.append($n$)
        end for
    end for
    return $I$
end procedure
```

Figure 5.8

Document-At-A-Time

```
procedure DocumentAtATimeRetrieval($Q$, $I$, $f$, $g$, $k$)
    $L \leftarrow$ Array()
    $R \leftarrow$ PriorityQueue($k$)
    for all terms $w_i$ in $Q$ do
        $l_i \leftarrow$ InvertedList($w_i$, $I$)
        $L$.add( $l_i$ )
    end for
    for all documents $d \in I$ do
        $s_d \leftarrow 0$
        for all inverted lists $l_i$ in $L$ do
            if $l_i$.getCurrentDocument() $= d$ then
                $s_d \leftarrow s_d + g_i(Q) f_i(l_i)$        ▷ Update the document score
            end if
            $l_i$.movePastDocument( $d$ )
        end for
        $R$.add( $s_d, d$ )
    end for
    return the top $k$ results from $R$
end procedure
```

$$\boxed{R(Q,D) = \sum_i g_i(Q) f_i(D)}$$

Figure 5.16

Term-At-A-Time

```
procedure TermAtATimeRetrieval($Q$, $I$, $f$, $g$ $k$)
    $A \leftarrow$ HashTable()
    $L \leftarrow$ Array()
    $R \leftarrow$ PriorityQueue($k$)
    for all terms $w_i$ in $Q$ do
        $l_i \leftarrow$ InvertedList($w_i$, $I$)
        $L$.add( $l_i$ )
    end for
    for all lists $l_i \in L$ do
        while $l_i$ is not finished do
            $d \leftarrow l_i$.getCurrentDocument()
            $A_d \leftarrow A_d + g_i(Q)f(l_i)$
            $l_i$.moveToNextDocument()
        end while
    end for
    for all accumulators $A_d$ in $A$ do
        $s_d \leftarrow A_d$                    ▷ Accumulator contains the document score
        $R$.add( $s_d, d$ )
    end for
    return the top $k$ results from $R$
end procedure
```

$$\boxed{R(Q,D) = \sum_i g_i(Q) f_i(D)}$$

Figure 5.18

Reminder about using lists

- "Align" inverted lists / traverse in parallel
- Allows
  - Simple Boolean AND, OR, NOT
  - Proximity operators (if has position information)
  - Combining scores across terms

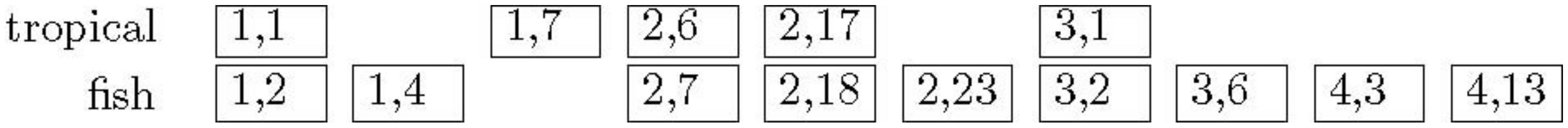

Figure 5.6

Conjunctive Term-at-a-Time

Figure 5.20

```
procedure TermAtATimeRetrieval($Q$, $I$, $f$, $g$, $k$)
    $A \leftarrow$ Map()
    $L \leftarrow$ Array()
    $R \leftarrow$ PriorityQueue($k$)
    for all terms $w_i$ in $Q$ do
        $l_i \leftarrow$ InvertedList($w_i$, $I$)
        $L$.add( $l_i$ )
    end for
    for all lists $l_i \in L$ do
        $d_0 \leftarrow -1$
        while $l_i$ is not finished do
            if $i = 0$ then
                $d \leftarrow l_i$.getCurrentDocument()
                $A_d \leftarrow A_d + g_i(Q) f(l_i)$
                $l_i$.moveToNextDocument()
            else
                $d \leftarrow l_i$.getCurrentDocument()
                $d' \leftarrow A$.getNextAccumulator($d$)
                $A$.removeAccumulatorsBetween($d_0, d'$)
                if $d = d'$ then
                    $A_d \leftarrow A_d + g_i(Q) f(l_i)$
                    $l_i$.moveToNextDocument()
                else
                    $l_i$.skipForwardToDocument($d'$)
                end if
                $d_0 \leftarrow d'$
            end if
        end while
    end for
    for all accumulators $A_d$ in $A$ do
        $s_d \leftarrow A_d$                    ▷ Accumulator contains the document score
        $R$.add( $s_d, d$ )
    end for
    return the top $k$ results from $R$
end procedure
```

Conjunctive Document-at-a-Time

Figure 5.21

```
procedure DocumentAtATimeRetrieval($Q$, $I$, $f$, $g$, $k$)
    $L \leftarrow$ Array()
    $R \leftarrow$ PriorityQueue($k$)
    for all terms $w_i$ in $Q$ do
        $l_i \leftarrow$ InvertedList($w_i$, $I$)
        $L$.add( $l_i$ )
    end for
    $d \leftarrow -1$
    while all lists in $L$ are not finished do
        $s_d \leftarrow 0$
        for all inverted lists $l_i$ in $L$ do
            if $l_i$.getCurrentDocument() $> d$ then
                $d \leftarrow l_i$.getCurrentDocument()
            end if
        end for
        for all inverted lists $l_i$ in $L$ do
            $l_i$.skipForwardToDocument($d$)
            if $l_i$.getCurrentDocument() $= d$ then
                $s_d \leftarrow s_d + g_i(Q) f_i(l_i)$            ▷ Update the document score
                $l_i$.movePastDocument( $d$ )
            else
                $d \leftarrow -1$
                break
            end if
        end for
        if $d > -1$ then $R$.add( $s_d, d$ )
        end if
    end while
    return the top $k$ results from $R$
end procedure
```