Search Algorithms for Speech Recognition

Berlin Chen 2004
References

• Books

• Papers
Introduction

• Template-based: **without statistical modeling/training**
  – Directly compare/align the testing and reference waveforms on their features vector sequences (with different length, respectively) to derive the overall distortion between them
  – **Dynamic Time Warping (DTW)**: warp speech templates in the time dimension to alleviate the distortion

• Model-based: **HMM are using for recognition systems**
  – Concatenate the subword models according to the pronunciation of the words in a lexicon
  – The states in the HMM can be expanded to form the state-search space (HMM state transition network) in the search
  – Apply appropriate search strategies
Template-based Speech Recognition

• Dynamic Time Warping (DTW) is simple to implement and fairly effective for small-vocabulary Isolated word speech recognition
  – Use **dynamic programming (DP)** to temporally align patterns to account for differences in speaking rates across speakers as well as across repetitions of the word by the same speakers

• Drawback
  – Do not have a principled way to derive an averaged template for each pattern from a large training samples
  – A multiplicity of reference templates is required to characterize the variation among different utterances
Template-based Speech Recognition (cont.)

- Example

\[ D_{\min}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} D_{\min}[i_k, j_k][i_{k-1}, j_{k-1}] \]

\[ = \min_{i_{k-1}, j_{k-1}} D_{\min}(i_{k-1}, j_{k-1}) + d[i_k, j_k][i_{k-1}, j_{k-1}] \]

( where \( D_{\min}[i_k, j_k][i_{k-1}, j_{k-1}] = D_{\min}(i_{k-1}, j_{k-1}) + d[i_k, j_k][i_{k-1}, j_{k-1}] \) )
Model-based Speech Recognition

- A search process to uncover the word sequence 
  \( \hat{W} = w_1, w_2, ..., w_m \) that has the maximum posterior 
  probability \( P(W|X) \)

\[
\hat{W} = \arg \max_w P(W|X)
\]

\[
= \arg \max_w \frac{P(W)P(X|W)}{P(X)}
\]

\[
= \arg \max_w P(W)P(X|W)
\]

\( W = w_1, w_2, ..., w_i, ..., w_m \)

where \( w_i \in V : \{v_1, v_2, ..., v_N\} \)

Language Model Probability

Acoustic Model Probability

N-gram Language Modeling

Unigram:
\[
P(w_1w_2...w_i) \approx P(w_i)P(w_2)...P(w_i), \quad P(w_i) = \frac{C(w_i)}{\Sigma C(w_i)}
\]

Bigram:
\[
P(w_1w_2...w_i) \approx P(w_i)P(w_2|w_i)...P(w_i|w_{i-1}), \quad P(w_i|w_{i-1}) = \frac{C(w_i|w_{i-1})}{C(w_{i-1})}
\]

Trigram:
\[
P(w_1w_2...w_i) \approx P(w_i)P(w_2|w_i)P(w_3|w_2w_i)...P(w_i|w_{i-2}w_{i-1}), \quad P(w_i|w_{i-2}w_{i-1}) = \frac{C(w_i|w_{i-2}w_{i-1})}{C(w_{i-2}w_{i-1})}
\]
Model-based Speech Recognition (cont.)

- Therefore, the model-based continuous speech recognition is both a pattern recognition and search problems
  - The acoustic and language models are built upon a statistical pattern recognition framework
  - In speech recognition, making a search decision is also referred to as a decoding process (or a search process)
    - Find a sequence of words whose corresponding acoustic and language models best match the input signal
    - The search space (complexity) is highly imposed by the language models
- The model-based continuous speech recognition is usually with the Viterbi (plus beam, or Viterbi beam) search or A* stack decoders
  - The relative merits of both search algorithms were quite controversial in the 1980s
Model-based Speech Recognition (cont.)

- Simplified Block Diagrams
Basic Search Algorithms
What Is “Search”?

• What Is “Search”: *moving around, examining things, and making decisions about whether the sought object has yet been found*
  – Classical problems in AI: *traveling salesman’s problem, 8-queens, etc.*

• The directions of the search process
  – Forward search (reasoning): from initial state to goal state(s)
  – Backward search (reasoning): from goal state(s) to goal state
  – Bidirectional search
    • Seems particular appealing if the number of nodes at each step grows exponential with the depth that need to be explored
What Is “Search”? (cont.)

• Two categories of search algorithms
  – Uninformed Search (Blind Search)
    • Depth-First Search
    • Breadth-First Search

  *Have no sense of where the goal node lies ahead!*

  – Informed Search (Heuristic Search)
    • A* search (Best-First Search)

  *The search is guided by some domain knowledge (or heuristic information)! (e.g. the predicted distance/cost from the current node to the goal node)*

  – Some heuristic can reduce search effort without sacrificing optimality
Depth-First Search

• The deepest nodes are expanded first and nodes of equal depth are ordered arbitrary
• Pick up an arbitrary alternative at each node visited
• Stick with this partial path and walks forward from the partial path, other alternatives at the same level are ignored completely
• When reach a dead-end, go back to last decision point and proceed with another alternative

• *Depth-first search could be dangerous because it might search an impossible path that is actually an infinite dead-end*
Breadth-First Search

• Examine all the nodes on one level before considering any of the nodes on the next level (depth)
• Breadth-first search is guaranteed to find a solution if one exists
  – But it might not find a short-distance path, it’s guaranteed to find one with few nodes visited (minimum-length path)
• Could be inefficient

Implemented with a FIFO queue
A* search

• History of A* Search in AI
  – The most studied version of the best-first strategies (Hert, Nilsson, 1968)
  – Developed for additive cost measures (The cost of a path = sum of the costs of its arcs)

• Properties
  – Can sequentially generate multiple recognition candidates
  – Need a good heuristic function

• Heuristic
  – A technique (domain knowledge) that improves the efficiency of a search process
  – Inaccurate heuristic function results in a less efficient search
  – The heuristic function helps the search to satisfy admissible condition

• Admissibility
  – The property that a search algorithm guarantees to find an optimal solution, if there is one
A* search

• A Simple Example
  – **Problem**: Find a path with highest score from root node “A” to some leaf node (one of “L1”, “L2”, “L3”, “L4”)

\[ f(n) = g(n) + h(n) \], evaluation function of node \( n \)

\( g(n) \): cost from root node to node \( n \), decoded partial path score

\( h^*(n) \): exact score from node \( n \) to a specific leaf node

\( h(n) \): estimated score from node \( n \) to goal state, heuristic function

Admissibility: \( h(n) \geq h^*(n) \)
A* search (cont.)

- **A Simple Example:**
  
  Evaluation function of node $n$:
  \[ f(n) = g(n) + h(n) \]

  ![Diagram](image.png)

<table>
<thead>
<tr>
<th>Node</th>
<th>$g(n)$</th>
<th>$h(n)$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>F</td>
<td>7</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>11</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>L1</td>
<td>9</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>L2</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>L3</td>
<td>12</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>L4</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

**Proving the Admissibility of A* Algorithm:**

Suppose when algorithm terminates, “$G$” is a complete path on the top of the stack and “$p$” is a partial path which presents somewhere on the stack. There exists a complete path “$P$” passing through “$p$”, which is not equal to “$G$” and is optimal.

Proof:

1. “$P$” is a complete which passes through “$p$”, $f(P) <= f(p)$
2. Because “$G$” is on the top of the stack, $f(G) >= f(p) >= f(P)$
3. Therefore, it makes contrariety!!
A* search: Exercises

- Please find a path from the initial state $\alpha$ to one of the four goal states ($\beta_1$, $\beta_2$, $\beta_3$, $\beta_4$) with the shortest path cost. Each arc is associated with a number representing its corresponding cost to be taken, while each node is associated with a number standing for the expected cost (the heuristic score/function) to one of the four goal states.
A* search: Exercises (cont.)

• Problems
  – What is the first goal state found by the depth-first search, which always selects a node’s left-most child node for path expansion? Is it an optimal solution? What is the total search cost?
  – What is the first goal state found by the bread-first search, which always expends all child nodes at the same level from left to right? Is it an optimal solution? What is the total search cost?
  – What is the first goal state found by the A* search using the path cost and heuristic function for path expansion? Is it an optimal solution? What is the total search cost?
  – What is the search path cost if the A* search was used to sequentially visit the four goal states?
Beam Search

- A widely used search technique for speech recognition systems
  - It’s a breadth-first search and progresses along with the depth
  - Unlike traditional breadth-first search, beam search only expands nodes that are likely to succeed at each level
    - Keep up to m-best nodes at each level (stage)
    - Only these nodes are kept in the beam, the rest are ignored (pruned)

*Figure 12.9* Beam search for the city-travel problem. The nodes with gray color are the ones kept in the beam. The transparent nodes were explored but pruned because of higher cost. The dotted nodes indicate all the savings because of pruning [42].
Beam Search (cont.)

- Used to prune unlikely paths in recognition task
- Need some criteria (hypotheses) to prune paths
Fast-Match Search

• Two Stage Processing
  – First stage: use a simplified grammar network (or acoustic models) to generate N likely words
    • Lower order language models, CI acoustic models
  – Second stage: use a precise grammar network to reorder these N words

The fast-match algorithm paradigm
Review:
Search Within a Given HMM
Calculating the Probability of an Observation Sequence on an HMM Model

- Direct Evaluation: **without using recursion (DP, dynamic programming)** and memory

\[ P(O|\lambda) = \sum_{s} P(s|\lambda)P(O|s,\lambda) \]
\[ = \sum_{s} \left[ \pi_{s_1}a_{s_1s_2}a_{s_2s_3}...a_{s_{T-1}s_T} \right]b_{s_1}(o_1)b_{s_2}(o_2)...b_{s_T}(o_T) \]
\[ = \sum_{s_1,s_2,...,s_T} \pi_{s_1}b_{s_1}(o_1)a_{s_1s_2}b_{s_2}(o_2)...a_{s_{T-1}s_T}b_{s_T}(o_T) \]

- Huge Computation Requirements: \( O(N^T) \)
  - Exponential computational complexity
  - **Complexity**: \((2T-1)N^T \text{ MUL } \approx 2TN^T, N^T-1 \text{ ADD}\)

- A more efficient algorithms can be used to evaluate
  - **Forward/Backward Procedure/Algorithm**
Calculating the Probability of an Observation Sequence on an HMM Model (cont.)

• Forward Procedure
  – Base on the HMM assumptions, the calculation of $P\left(s_t | s_{t-1}, \lambda \right)$ and $P\left(o_t | s_t, \lambda \right)$ involves only $s_{t-1}$, $s_t$ and $o_t$, so it is possible to compute the likelihood $P(O|\lambda)$ with recursion on $t$

  – Forward variable: $\alpha_t(i) = P\left(o_1o_2...o_t, s_t = i | \lambda \right)$
    • The probability that the HMM is in state $i$ at time $t$ having generating partial observation $o_1o_2...o_t$
Calculating the Probability of an Observation Sequence on an HMM Model (cont.)

- **Forward Procedure (Cont.)**
  - **Algorithm**
    1. Initialization $\alpha_t(i) = \pi_i b_i(o_i), 1 \leq i \leq N$
    2. Induction $\alpha_{t+1}(j) = \left[ \sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1}), 1 \leq t \leq T-1, 1 \leq j \leq N$
    3. Termination $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

- **Complexity:** $O(N^2T)$
  - MUL: $N(N+1)(T-1) + N \approx N^2T$
  - ADD: $(N-1)N(T-1) \approx N^2T$

- **Based on the lattice (trellis) structure**
  - Computed in a *time-synchronous* fashion from *left-to-right*, where each cell for time $t$ is completely computed before proceeding to time $t+1$
  - All state sequences, regardless how long previously, merge to $N$ nodes (states) at each time instance $t$
Calculating the Probability of an Observation Sequence on an HMM Model (cont.)

• Backward Procedure

  – Backward variable: \( \beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T | s_t = i, \lambda) \)

  – Algorithm
  1. Initialization \( \beta_1(i) = 1, 1 \leq i \leq N \)
  2. Induction \( \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), 1 \leq t \leq T - 1, 1 \leq j \leq N \)
  3. Termination \( P(O | \lambda) = \sum_{j=1}^{N} b_j(o_1) \beta_1(j) \)

  – Complexity: \( O(N^2T) \)

Complexity MUL: \( 2N^2(T - 1) \approx N^2T \); ADD: \( (N - 1)N(T - 1) \approx N^2T \)

\[
P(O, q_t = i | \lambda) \\
= P(O, q_t = i, \lambda) / P(\lambda) \\
= P(O | q_t = i, \lambda) \cdot P(q_t = i, \lambda) / P(\lambda) \\
= P(O | q_t = i, \lambda) P(q_t = i | \lambda) \\
= P(o_1, o_{t+2}, \ldots, o_T | q_t = i, \lambda) P(q_t = i | \lambda) \quad \text{[\because \text{observation independence}]} \\
= P(o_{t+1}, o_{t+2}, \ldots, o_T | q_t = i, \lambda) P(o_1, o_2, \ldots, o_t | q_t = i, \lambda) P(q_t = i | \lambda) \\
= P(o_{t+1}, o_{t+2}, \ldots, o_T | q_t = i, \lambda) P(o_1, o_2, \ldots, o_t, q_t = i | \lambda) \\
= \alpha_t(i) \beta_t(i)
\]
Choosing an Optimal State Sequence $S=(s_1, s_2, \ldots, s_T)$ on an HMM Model

• **Viterbi Algorithm**
  - The Viterbi algorithm can be regarded as the dynamic programming algorithm applied to the HMM or as a modified forward algorithm
    • Instead of summing up probabilities from different paths coming to the same destination state, the Viterbi algorithm picks and remembers the best path
    • Find a single optimal state sequence $S=(s_1, s_2, \ldots, s_T)$
  - The Viterbi algorithm also can be illustrated in a trellis framework similar to the one for the forward algorithm
Choosing an Optimal State Sequence
\( S = (s_1, s_2, \ldots, s_T) \) on an HMM Model (cont.)

• Viterbi Algorithm (Cont.)
  – Algorithm

Find a best state sequence \( S = (s_1, s_2, \ldots, s_T) \) for a given observation \( O = (o_1, o_2, \ldots, o_T) \)?

1. Define a new variable

\[
\delta_t(i) = \max_{s_1, s_2, \ldots, s_{t-1}} P[s_1, s_2, \ldots, s_{t-1}, s_t = i, o_1, o_2, \ldots, o_t | \lambda]
\]

= the best score along a single path at time \( t \), which accounts for the first \( t \) observation and ends in state \( i \)

2. By induction \( \therefore \) \( \delta_{t+1}(j) = \max_{1 \leq i \leq N} \delta_t(i)a_{ij}b_j(o_{t+1}) \)

\( \psi_{t+1}(j) = \arg \max_{1 \leq i \leq N} \delta_t(i)a_{ij} \) .... For backtracing

3. We can backtrace from \( s_T^* = \arg \max_{1 \leq i \leq N} \delta_T(i) \)

– Complexity: \( O(N^2T) \)
Choosing an Optimal State Sequence
\( S = (s_1, s_2, \ldots, s_T) \) on an HMM Model (cont.)

- **Viterbi Algorithm (Cont.)**
  - In practice, we calculate the logarithmic value of a given state sequence instead of its real value

1. Define
   \[
   \delta_t(i) = \max_{s_1, s_2, \ldots, s_{t-1}} \log P[s_1, s_2, \ldots, s_{t-1}, s_t = i, o_1, o_2, \ldots, o_t | \lambda]
   \]
   = the best log score along a single path at time \( t \), which accounts for the first \( t \) observation and ends in state \( i \)

By induction:
\[
\delta_{t+1}(j) = \max_{1 \leq i \leq N} \left( \delta_t(i) + \log a_{ij} \right) + \log b_j(o_{t+1})
\]

\[
\psi_t(j) = \arg \max_{1 \leq i \leq N} \delta_{t-1}(i) + \log a_{ij} \quad \text{.....For backtracing}
\]

We can backtrace from
\[
q^*_t = \arg \max_{1 \leq i \leq N} \delta_t(i)
\]
Search in the HMM Networks
Digit/Syllable Recognition

• One-stage Search
  – Unknown number of digits/syllables
  – Search over a 3-dim grid

  – At each frame iteration, the maximum value achieved from the end states of all models in previous frame will be propagated and used to compete for the values of the start states of all models
  – May result with substitutions, deletions and insertions
Digit/Syllable Recognition (cont.)

- **Level-Building**
  - Known number of digits/syllables
  - Higher computation complexity, **no deletions and insertions**
  - Number of levels: number of digits in an utterance
  - Transitions from the last states of the previous models (previous level) to the first states of specific models (current level)
Isolated Word Recognition

- Word boundaries are known (after endpoint detection)
- Two search structures
  - **Lexicon-List** (Linear Lexicon)
    - Each word is individually represented as a huge composite HMM by concatenating corresponding subword-level (phone/Initial-Final/syllable) HMMs
    - No sharing of computation between words when performing search
    - The search becomes a simple pattern recognition problem, and the word with the highest forward or Viterbi probability is chosen as the recognition word
  - **Tree Structure** (Tree Lexicon)
    - Arrange the subword-level (phone/Initial-Final/syllable) representations of the words in vocabulary into a tree structure
    - Each arc stands for an HMM or subword-level modeling
    - Sharing of computation between word as much as possible
Isolated Word Recognition (cont.)

- Two search structures (Cont.)

Linear lexicon  18 arcs

Tree lexicon  13 arcs
Isolated Word Recognition (cont.)

• More about the Tree Lexicon
  – The idea of using a tree represented was already suggested in 1970s in the CASPERS system and the LAFS system
  – When using such a lexical tree in a language model (bigram or trigram) and dynamic programming, there are technical details that have to taken into account and require a careful structuring of the search space (especially for continuous speech recognition to be discussed later)
    • Delayed application of language model until reaching tree leaf nodes
    • A copy of the lexical tree for each alive language model history in dynamic programming for continuous speech recognition
Continuous Speech Recognition (CSR)

- CSR is rather complicated, since the search algorithm has to consider the possibility of each word starting at arbitrary time frame
- Linear Lexicon Without Language Modeling
Continuous Speech Recognition (cont.)

• Linear Lexicon With Unigram Language Modeling

*Figure 12.14* A unigram grammar network where the unigram probability is attached as the transition probability from starting state $S$ to the first state of each word HMM.
Continuous Speech Recognition (cont.)

- Linear Lexicon With Bigram Language Modeling

Figure 12.15 A bigram grammar network where the bigram probability $P(w_j | w_i)$ is attached as the transition probability from word $w_i$ to $w_j$ [19].

Figure 12.16 Reducing bigram expansion in a search by using the backoff node. In addition to normal bigram expansion arcs for all observed bigrams, the last state of word $w_j$ is first connected to a central backoff node with transition probability equal to backoff weight $\alpha(w_j)$. The backoff node is then connected to the beginning of each word $w_j$ with its corresponding unigram probability $P(w_j)$ [12].
Continuous Speech Recognition (cont.)

- Linear Lexicon With Trigram Language Modeling

![Diagram of a trigram grammar network](image)

*Figure 12.17* A trigram grammar network where the trigram probability $P(w_i | w_j, w_k)$ is attached to transition from grammar state word $w_j$ to the next word $w_i$. Illustrated here is a two-word vocabulary, so there are four grammar states in the trigram network [19].
Further Studies on Implementation Techniques for Speech Recognition
Isolated Word Recognition
Search Strategy: Beam search

- **Tree Structure for Pronunciation Lexicon**
- **Initialization for Dynamic Programming**
- **Two-Level Dynamic Programming**
  - **Within HMM**
  - **Between HMMs** (Arc extension)
Isolated Word Recognition
Search Strategy: A* Search

• Applied to Mandarin Isolated Word Recognition
  – Forward Trellis Search (Heuristic Scoring)
    • A forward time-synchronous Viterbi-like trellis search for generating the heuristic score
    • Using a simplified grammar network of different degree grammar type: (Over-generated Grammar)
      – No grammar
      – Syllable-pair grammar
      – No grammar with string length constraint grammar
    • Syllable-pair with string length constraint grammar
  – Backward A* Tree Search
    • A backward time-asynchronous viterbi-like A* tree search for finding the “exact” word
    • A backward syllabic tree without overgenerating the lexical vocabulary
Isolated Word Recognition
Search Strategy: A* Search (cont.)

Grammar Networks for Heuristic Scoring

Four types of simplified grammar networks used in the tree search.
Isolated Word Recognition
Search Strategy: A* Search (cont.)

Steps in A* Search:
- At each iteration of the algorithm:
  - A sorted list (or stack) of partial paths, each with an evaluation function
  - The partial path with the highest evaluation function -
  - Expanded
  - For each one-phone (or one syllable or one arc) extensions permitted by the lexicon, the evaluation functions of the extended paths are calculated
  - And the extended partial paths are inserted into the stack at the appropriate position (sorted according to "evaluation function")
- The algorithm terminates -
  - When a complete path (or word) appears on the top of the stack
Keyword Spotting

• The Common Aspect of Most Word Spotting Applications
  – It is only necessary to extract partial information from the input speech utterance
  – Many automated speech recognition problems can be loosely described by this requirement
    • Speech message browsing
    • Command spotting
    • Telecommunications services (applications)

Hesitation,
Repetition,
Out-of-vocabulary words (OOV)

“Mm,...,”
“I wanna talk ..talk to..”
“What?”
“幫我找台..台灣銀行的ㄟ電話”
Keyword Spotting (cont.)

• General Framework of Keyword Spotting
  – Viterbi Decoding (Continuous Speech Recognition)
  – Utterance Verification (a two-stage approach)

A simple, unconstrained finite state network contains $N$ keywords and $M$ fillers. Associated with each keyword and filler are word transition penalties.

A continuous stream of keywords and fillers.
Keyword Spotting (cont.)

- Single-keyword Spotting

![Diagram of keyword spotting]

- DeltaW(S_{W}, T-1)
- DeltaF(S_{F_2}, T-1)
- Max{ DeltaF(S_{F_2}, T-1), DeltaW(S_{W}, T-1) }
- Prob.=1.0
- DeltaF1(S_{F_1}, t-1)
- DeltaW(S_{W}, t-1)
- DeltaF2(S_{F_2}, t-1)
- DeltaF2(1, t-1)
- DeltaF2(1, t)
- DeltaW(S_{W}, t)
- DeltaF1(1, t-1)
- DeltaF1(1, t)
- Prob.=1.0
Keywords Spotting (cont.)

**Case Study: A* search for Mandarin Keyword Spotting**

- **Search Framework**
  - **Forward Heuristic Scoring**

\[
 f(t) = a \cdot s(t) + b \cdot s(t) + (1-a-b) \cdot f(t)
\]

\[
 h^*(n_k, t) = \text{MAX} \left[ f_L(t_1) + h(n_k, t_1 + 1, t) \right]
\]

The structure of the compact syllable lattice and the filler models in the first pass
**Case Study:** A* search for Mandarin Keyword Spotting

- **Search Framework**
  - Backward Time-Asynchronous A* Search

\[
E_p(n_k) = \max_{0 < t < T} [d_p(n_k,t) + h^*(n_k,t-1)]
\]
\[
d_p(n_k,t) = \max_{t < t_2 < T} [g_p(n_k,t_2) - 1] + f_R(t_2)
\]

The search framework of key-phrase spotting
Data Structure for the Lexicon Tree

• Trie Structure

```c
struct DEF_LEXICON_TREE {
    short Model_ID;
    short WD_NO;
    int *WD_ID;
    int Leaf;
    struct Tree *Child;
    struct Tree *Brother;
    struct Tree *Father;
};
```