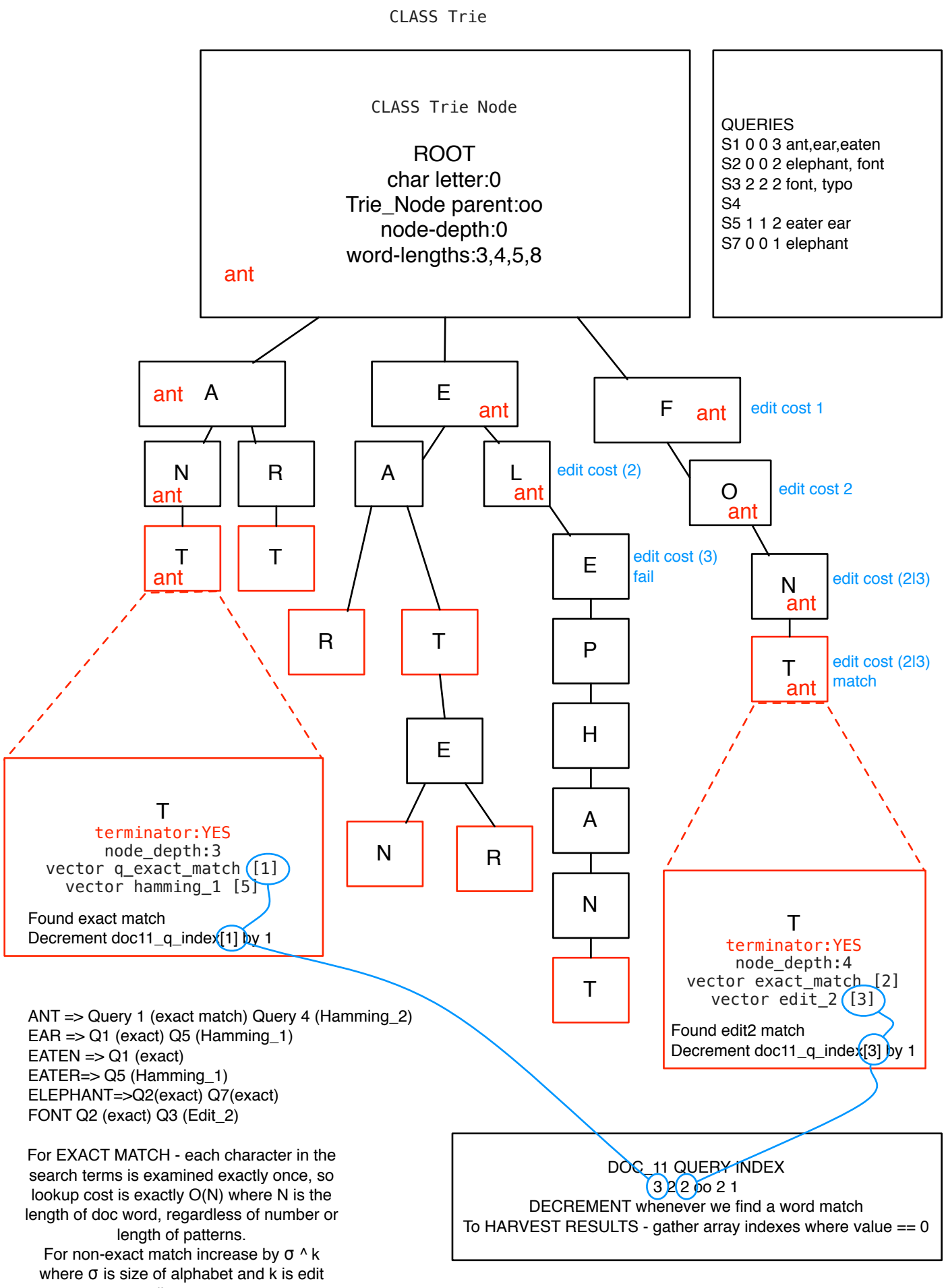
<< incoming document << id 11 count 130 http dbpedia resource remote virtual ant font eaten...




```
CLASS Trie Node
    char letter;
    Tie Node parent;
    int depth;
     char* [26] child_alphabet;   //initialise with 26 elements val oo
     vetor <int>word_lengths;
      bool terminator
        vector <int> q_exact_match   //query_ids of exact_match queries containting this word
        vector <int> q_hamming_1    //query_ids of hamming_match_1 queries containing this word
        vector <int> q_hamming_2   //query_ids of hamming_match_2 queries containing this word
        vector <int> q_edit_1   //query_ids of edit_distance_1 queries containing this word
        vector <int> q_edit_2   //query_ids of edit_distance_2 queries containing this word
    }
       vector <int> matching_docs  //do we need this?

       - void check_match_valid (query_id, match_type) {
         // look up query_Id in list of eliminated queries
         // if it is there, delete it from match_type
         // this way we don't need to delete eliminated entries.
         // OPTIMISATION=>may need to reconsider for deleting eliminated entries.
       }

       -void match_found (doc_id, query_id) {
            decrement the doc_id query_index
       }

       -void add_char (query_char, query_id, match_type, match_dist) {
       //adding the next character from a new query word;

            if (query_char == \0) {  //we've reached the end for this query word
                //reached word termination
                switch (match_type){
                   case exact_match:
                    push_back(q_exact_match, query_id);
                   break;
                   case hamming_match:
                     if (match_dist == 1) push_back (q_hamming_1, query_id)
                     else push_back (q_hammning_2, query_id)
                   break;
                   case edit_distance:
                     if (match_dist == 1) push_back (q_edit_1, query_id)
                     else push_back (q_edit_2, query_id)
                   break;
                }

            } else {  // not yet terminating

                if (child_alphabet[char]==oo) {
                    new Trie Node [query_char]
                }
                TrieNode* node = child_alphabet[char];

                node::add_char (query_char+1, query_id, match_type, match_dist)
            }
       }

       - void log_node_depths {
       //when we have reached the final insert of a new query_word,
       //only then do we know our word length.
       //Npw climb back up the tree to register word_length with every parent node.
       //we use this info at every node to work out whether to explore the tree further
       //(eg if doc-word length is 8 but following this node only yields
       //query words max 6 length, don't bother)
           parent add_word_length[self.depth]

       }
       - int  depth {
       if (!_depth) {
          _depth = parent.depth+1;
        }
        return _depth;
       }
}
```