package com.twitter.search.earlybird.queryparser;

import java.util.ArrayList;

import java.util.List;

import java.util.Set;

import com.google.common.collect.Sets;

/\*\*

\* Used to store information relevant to processing query groups for HighFrequencyTermPairExtractor

\* and HighFrequencyTermPairRewriter

\*/

public class HighFrequencyTermQueryGroup {

protected final int groupIdx;

protected final int parentGroupIdx;

// The number of nodes in this group.

protected int numMembers = 0;

// For the rewrite visitor: Incremented once at the end of each of this group's nodes' visits.

protected int numVisits = 0;

// The set of tokens that should be removed from the query if seen as an individual term and

// rewritten in the query as a hf term pair.

protected final Set<String> hfTokens = Sets.newTreeSet();

// Tokens that can be used to restrict searches but should not be scored. They will be given a

// weight of 0.

protected final Set<String> preusedHFTokens = Sets.newTreeSet();

// Set of phrases that should be removed from the query if seen as an individual phrase and

// rewritten in the query as a hf term phrase pair.

protected final Set<String> hfPhrases = Sets.newTreeSet();

// Phrases that can be used to restrict searches but should not be scored. They will be given a

// weight of 0.

protected final Set<String> preusedHFPhrases = Sets.newTreeSet();

// The first found hf\_term, or the hf\_term of an ancestor with the same isPositive value.

protected String distributiveToken = null;

// If it is a single node group, isPositive is true iff that node is true.

// Otherwise, isPositive is false iff the root of the group is a disjunction.

protected final boolean isPositive;

public HighFrequencyTermQueryGroup(int groupIdx, boolean positive) {

this(groupIdx, -1, positive);

}

public HighFrequencyTermQueryGroup(int groupIdx, int parentGroupIdx, boolean positive) {

this.groupIdx = groupIdx;

this.parentGroupIdx = parentGroupIdx;

isPositive = positive;

}

public boolean hasPhrases() {

return !hfPhrases.isEmpty() || !preusedHFPhrases.isEmpty();

}

protected List<String> tokensFromPhrases() {

if (!hasPhrases()) {

return null;

}

List<String> tokens = new ArrayList<>();

for (String phrase : hfPhrases) {

for (String term : phrase.split(" ")) {

tokens.add(term);

}

}

for (String phrase : preusedHFPhrases) {

for (String term : phrase.split(" ")) {

tokens.add(term);

}

}

return tokens;

}

protected void removePreusedTokens() {

hfTokens.removeAll(preusedHFTokens);

List<String> phraseTokens = tokensFromPhrases();

if (phraseTokens != null) {

hfTokens.removeAll(phraseTokens);

preusedHFTokens.removeAll(phraseTokens);

}

hfPhrases.removeAll(preusedHFPhrases);

}

protected String getTokenFromPhrase() {

List<String> phraseTokens = tokensFromPhrases();

if (phraseTokens != null) {

return phraseTokens.get(0);

} else {

return null;

}

}

}