Introduction

A simple algorithm for intersections was constructed for this lab. The intersection algorithm has an important role in information retrieval, it is used to organize indexes and dictionaries to make biwords easier to find and is therefore essential in search engines. This lab combines test cases with skeleton algorithms to practically lead students to understand the structure and construction of intersection algorithms, which have far-reaching relevance for information processing and retrieving.

Problem analyzes

The problem of this lab is an algorithm for intersection is in need of completing, it should be able to find matches where the two positions are within k distance.

Based on the given test cases and posting classes, it can be understood that the desired algorithm must be able to retrieve two positions close to the value of k and return them together with the docID.

Consider introducing new variables and parameters to calibrate the positions and input k, using loops to achieve the intersection

Implementation details

First enter the length of the list and the required count variables i, j

len1 = size(p1);

len2 = size(p2);

int i = j = 0;

And then, with the guarantee that the length is not null, introduce the comparison

while (i != len1 and j != len2){ # while (p1 != nil and p2 != nil){

if (docID(p1[i]) == docID(p2[j])) {

l = [] ;

pp1 = position(p1[i]);

pp2 = position(p2[j]) ;

Compare the size of the position gap with k and decide whether to extract

plen1 = len(pp1);

plen2 = len(pp2);

ii = jj = 0;

while (ii != plen1) {

while (jj != plen2) {

if (abs(pp1[ii] - pp2[jj]) <= k) {

l.append(pp2[jj]);

}

else if (pp2[jj] > pp1[ii]) {

break jj+=1;

}

}

while (l != []and abs (l[0] - pp1[ii]) > k ){

l.remove(l[0]);

for (pp2[] in l) {

answer. Append([docID(p1[i]), pp1[ii], pp2[jj] ])

ii += 1 ;

i += 1 ;

j += 1;

}

else if docID(p1[i]) < docID(p2[j]) {

i += 1 ;

}else{

j += 1;

return answer

Conclusion

After successfully building the intersection algorithm, the docID and positions were successfully returned, which made finding biwords much easier. The main problem encountered in this experiment was the input and processing of the k and the processing of the positions, which was solved by using lists and if, while loops. The balance of performance and time consumption was also taken into account.

Thanks to the flexibility of the use of loops, this algorithm satisfies the functional requirements and to some extent balances the efficiency requirements, more complex algorithms could probably be built along with this technique in the future.