Siddaganga Institute of Technology, Tumakuru-5

(An Autonomous institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC with 'A Grade, Awarded Diamond College Rating by QS I-GAUGE and ISO 9001:2015 certified)

Information Retrieval Project

On

"Faster postings list intersection"

submitted *in the partial fulfillment of the requirements for V Semester,* ***Bachelor of Engineering*** in ***Artificial Intelligence & Data Science***

By

Srikanth B - 1SI20AD027

Sakshith raj - 1SI20AD023

Department of Computer Science & Engineering

Academic Year : 2022-23

Siddaganga Institute of Technology, Tumakuru-5

(An Autonomous institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC with 'A Grade, Awarded Diamond College Rating by QS I-GAUGE and ISO 9001:2015 certified)

Department of Computer Science & Engg.

Certificate

This is to certify that the Information Retrieval Project Report entitled **"Faster posting list intersection"** is a bonafide work carried out by **Srikanth B(1SI20AD023)** and **Sakshith Raj (1SI20AD023)** of V Semester B.E.(Artificial Intelligence & Data Science), Siddaganga Institute of Technology, during the academic year 2022-23.

Faculty Incharge
Dr. N R Sunitha
Professor
Dept. of CSE, SIT

Contents

Abstract:

Information retrieval is a vital component in today's world, where people rely on search engines and databases to find the information they need. One of the main challenges in information retrieval is the efficient intersection of posting lists. The posting list intersection is a critical operation where the system compares two or more lists to find the documents that are common to all of them.

To improve the efficiency of this operation, researchers have developed the technique of Skip Pointers. It involves storing extra pointers in the posting list that skip over large sections of the list that do not contain any common documents, reducing the number of comparisons needed during the intersection operation and thus reducing its time complexity. The skip pointers can be implemented using a fixed skip distance, dynamic skip distance or a combination of both, resulting in a significant improvement in the speed and efficiency of the intersection operation

The use of Skip Pointers has had a major impact on the field of information retrieval. It is a crucial technique for making information retrieval faster and more efficient. Researchers continue to explore new and innovative ways to improve the technique, such as combining fixed and dynamic skip distances to achieve an optimal solution. The implementation of Skip Pointers has helped to make information retrieval more accessible and user-friendly for people around the world.

Introduction to Information Retrieval:

Information retrieval (IR) is the science of searching and retrieving information from a collection of data, such as text documents or images, to satisfy an information need. It is a crucial aspect of modern-day life, as people are constantly looking for information and answers to their questions. With the vast amount of information available on the internet, IR has become even more important, as people expect quick and accurate results from their queries

The goal of IR is to provide users with the most relevant and accurate information in response to their queries. To achieve this, IR systems use various techniques such as keyword matching, content analysis, and machine learning algorithms. These techniques are used to index, store, and retrieve information from the collection of data, so that users can quickly access the information they need.

There are two main approaches to IR: Boolean retrieval and ranked retrieval. In Boolean retrieval, a user inputs a query consisting of keywords and logical operators, such as AND and OR. The IR system then returns a list of documents that match the query, where each document is either included or excluded based on the Boolean conditions. In ranked retrieval, the IR system returns a list of documents that match the user's query, but the results are ranked in order of relevance to the query. This approach is more sophisticated and takes into account various factors such as term frequency, inverse document frequency, and query context, to determine the relevance of each document.

The process of IR can be broken down into three main stages: pre-processing, indexing, and retrieval. In the pre-processing stage, the data collection is cleaned, normalized, and formatted for efficient processing. This includes removing duplicates, converting text to lowercase, and stemming or lemmatizing words to their base form. In the indexing stage, the IR system builds an index of the data collection, which is used to quickly retrieve information in response to queries. In the retrieval stage, the IR system uses the index to search the data collection and returns the most relevant results to the user.

One of the challenges of IR is the problem of information overload, where the data collection is so large that it is difficult for users to find the information they need. To address this issue, IR systems use techniques such as information filtering, which removes irrelevant information, and information visualization, which presents the results in a clear and organized manner.

In conclusion, information retrieval is a crucial aspect of modern-day life, as people rely on IR systems to find the information they need. IR systems use various techniques to search, store, and retrieve information from a collection of data, and the goal is to provide users with the most relevant and accurate information in response to their queries. With the vast amount of information available on the internet, IR continues to play a critical role in our lives, and researchers are constantly working to improve IR systems to better meet the needs of users.

Problem Statement:

The problem statement for faster postings list intersection using skip pointers is to develop a method for efficiently finding the common documents among multiple posting lists. The traditional method of intersecting posting lists has a time complexity of O(n1 + n2 + ... + nm), where n1, n2, ..., nm are the sizes of the posting lists. However, as the number of posting lists and the size of the collection increases, this method becomes increasingly slow and computationally expensive.

The proposed solution is to use skip pointers in the posting lists to reduce the number of comparisons that need to be made during the intersection operation. Skip pointers are additional pointers that are stored in the posting lists that allow the intersection algorithm to skip over large sections of the list that do not contain any common documents. This reduces the time complexity of the intersection operation and makes it faster and more efficient.

Objectives:

The main objectives of faster posting list intersection are to reduce the time and memory required for the intersection process, ensure scalability for handling large amounts of data, maintain accuracy in the results, and provide flexibility for different types of data and use cases. The intersection process is a critical step in information retrieval and its efficiency and effectiveness directly impact the overall performance of these applications. By optimizing the time and space efficiency, ensuring scalability, maintaining accuracy, and providing flexibility, faster posting list intersection can enable more effective and efficient information retrieval.

Design:

Algorithm:

```
INTERSECTWITHSKIPS(p1,p2)
   answer<-()
   while p1=NIL and p2=NIL
   do if docID(p1)=docID(p2)
         then ADD(answer,docID(p1))
           p1<-next(p1)
           p2<-next(p2)
     else if docID(p1)<docID(p2)
          then if hasSkip(p1)and(docID(skip(p1))<=docID(p2))
               then while hasSKip(p1)and(docID(skip(p1))<=docID(p2))
                      do p1<-skip(p1)
               else p1<-next(p1)
     else if hasSkip(p2)and(docID(skip(p2))<=docID(p1))
               then while hasSkip(p2)and(docID(skip(p2))<=docID(p1))
                      do p2<-skip(p2)
                else p2<-next(p2)
    return answer
```

This algorithm represents a method for performing an intersection between two posting lists using skip pointers. The input to the algorithm are two posting lists p1 and p2. The output is a result set which contains the document IDs that are present in both lists.

The algorithm operates in a loop that continues until both p1 and p2are equal to NIL (i.e., they have reached the end of their respective lists). In each iteration of the loop, the algorithm compares the document IDs of the current nodes in each list (docid(p1) and docid(p2)).If the document IDs are equal, the document ID is added to the result set (add(answer,docid(p1)), and both pointers are advanced to the next node in their respective lists (p1<-next(p1)) and p2<-next(p2)).

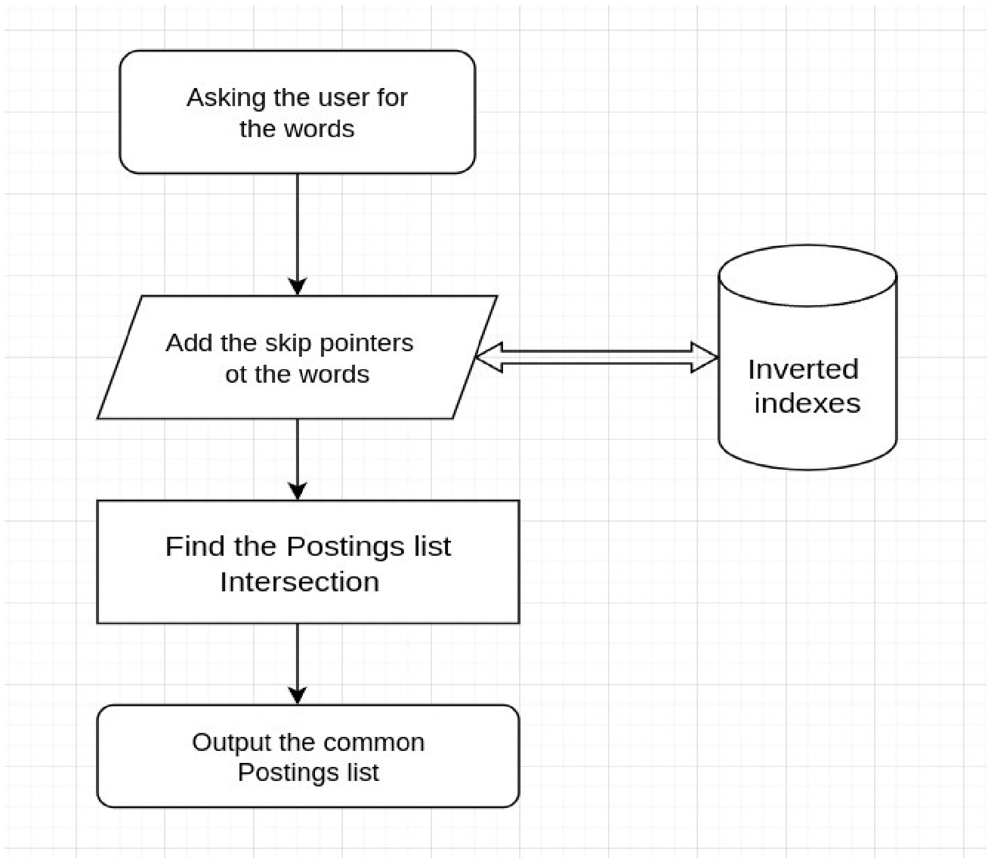
If the document ID of one list is smaller than the other, the algorithm checks if the current node has a skip pointer (hasSkip(p1) or hasSkip(p2)). If there is a skip pointer and the document ID of the skipped node is less than or equal to the document ID of the other list, the algorithm continues to follow the skip pointers until it reaches a node whose document ID is greater than the document ID of the other list.

If there is no skip pointer or the skip pointer leads to a node with a larger document ID, the algorithm advances to the next node in the corresponding list (p1<-next(p1) or p2<-next(p2)).

Finally, the result set answer is returned as the output of the algorithm.

};

Architecture diagram:


Asking the user for the words
Add the skip pointers ot the words
Inverted indexes
Find the Postings list Intersection
Output the common Postings list

Implementation:

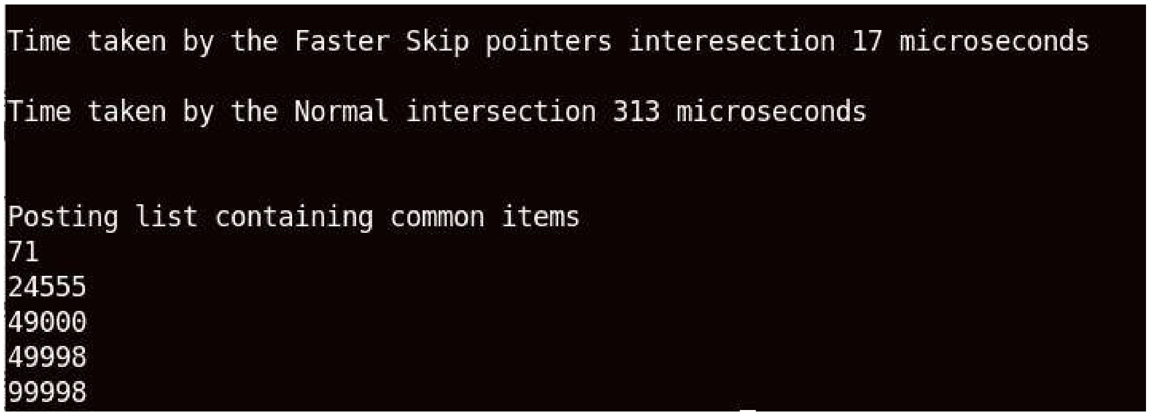
CODE:

```
#include <iostream>
#include <fstream>
#include <bits/stdc++.h>
using namespace std;
struct Node {
                    int val
                  Node* next;
                   Node* skip;
};
```

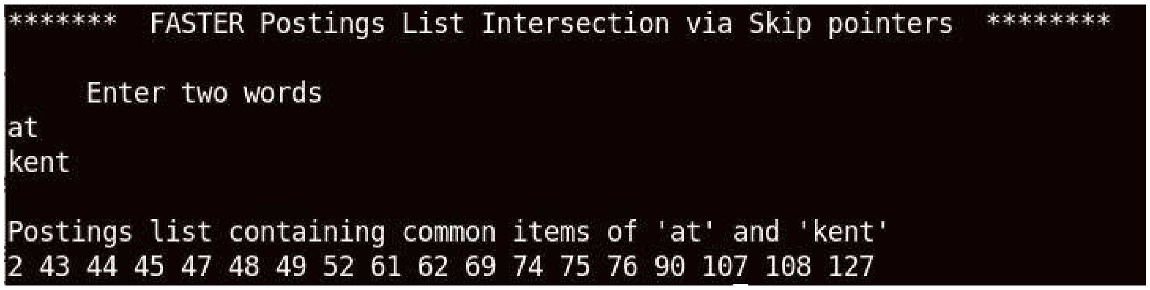
```
bool hasSkip(Node* node) {
          return node->skip != nullptr;
       }
Node* skip(Node* node) {
          return node->skip;
       }
Node* constructAndAddSkipPointers(const vector<int>& v) {
         Node* head = nullptr;
         Node* tail = nullptr;
         int n=v.size();
         int t=sqrt(n);
         Node* lastskip=nullptr;
         for (int i = 0; i < n ; i++)
         {
              Node* newNode = new Node({v[i], nullptr, nullptr});
              if (head == nullptr)
             {
                     head = newNode;
                     tail = newNode;
                     lastskip=newNode;
             }
               else
             {
                   tail->next = newNode
                   tail = newNode;
              }
               if ((i+1)%t == 0)
               {
                    lastskip->skip = tail;
                    lastskip=tail;
               }
          }
        return head;
  }
  vector<int> IntersectWithSkips(Node* p1, Node* p2) {
  vector<int> answer;
  while (p1 != nullptr && p2 != nullptr)
   {
          if (p1->val == p2->val)
          {
                    answer.push_back(p1->val);
                    p1 = p1->next;
                    p2 = p2->next;
           }
          else if (p1->val < p2->val)
          {
                   if (hasSkip(p1) && (skip(p1)->val <= p2->val))
                   {
                          while (hasSkip(p1) && (skip(p1)->val <= p2->val))
                          p1 = skip(p1);
                   }
                   else
                           p1 = p1->next;
           }
         else
         {
             if (hasSkip(p2) && (skip(p2)->val <= p1->val))
            {
                    while (hasSkip(p2) && (skip(p2)->val <= p1->val))
                           p2 = skip(p2);
             }
            else
                   p2 = p2->next;
           }
   }
     return answer;
}
```

```
map<string, set<int>> build_index(vector<string> files)            // building the inverted
index
{
        map<string, set<int>> inverted_index;
       for (int doc_id = 0; doc_id < files.size(); ++doc_id)             // reading the files
       {
              ifstream file(files[doc_id]);
              while (getline(file, line))
              {
                     stringstream ss(line);
                     while (ss >> word)
                     {                                                        {
                           transform(word.begin(), word.end(), word.begin(), ::tolower);
                                          inverted_index[word].insert(doc_id + 1);
                      }
              }
       }
        return inverted_index;
}

int main() {
       vector<string>  files ;
       for(int i=1;i<=128;i++)                  // reading the files name
       {
             string temp="/home/srikanth/Information retrieval/files/";
             string t=to_string(i);
             temp=temp+t+".txt";
             files.push_back(temp);
        }
   map<string,set<int>> inverted_index = build_index(files);
   string word1,word2;
   cout<<"***  FASTER Postings List Intersection via Skip pointers  ****\n"<<endl;
   cout<<"     Enter two words    "<<endl;
   cin>>word1>>word2;
   transform(word1.begin(), word1.end(), word1.begin(), ::tolower);
   transform(word2.begin(), word2.end(), word2.begin(), ::tolower);
   set<int> temp1=inverted_index[word1];
   set<int> temp2=inverted_index[word2];
   vector<int> p1(temp1.begin(),temp1.end());
   vector<int> p2(temp2.begin(),temp2.end());
   if(p1.size()==0 || p2.size()==0)
   {
          cout<<"The given word doesn't exist in the inverted index"<<endl;
          return 0;
   }
   Node * p11=constructAndAddSkipPointers(p1);
   Node * p22=constructAndAddSkipPointers(p2);
   vector<int> ans=IntersectWithSkips(p11,p22)
   cout<<"\nPostings list containing common items of '"<<word1<<"' and '
"<<word2<<"'"<<endl;
  for(int i=0;i<ans.size();i++)
          cout<<ans[i]<<" ";
  cout<<endl;

  return 0;
}
```

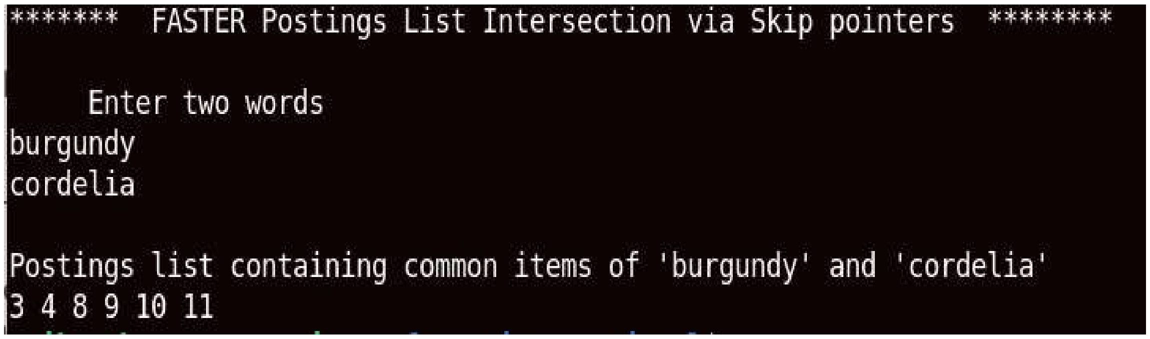
Output:



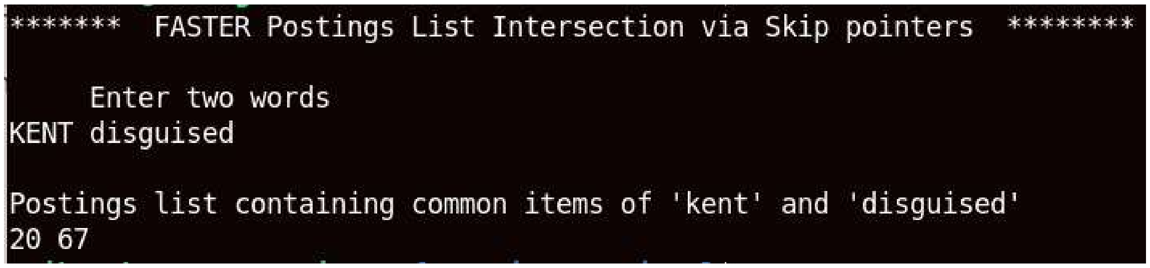
```
Time taken by the Faster Skip pointers interesection 17 microseconds

Time taken by the Normal intersection 313 microseconds


Posting list containing common items
71
24555
49000
49998
99998
```



```
*******  FASTER Postings List Intersection via Skip pointers  ********

     Enter two words
at
kent

Postings list containing common items of 'at' and 'kent'
2 43 44 45 47 48 49 52 61 62 69 74 75 76 90 107 108 127
```



```
*******  FASTER Postings List Intersection via Skip pointers  ********

     Enter two words
burgundy
cordelia

Postings list containing common items of 'burgundy' and 'cordelia'
3 4 8 9 10 11
```



```
*******  FASTER Postings List Intersection via Skip pointers  ********

     Enter two words
KENT disguised

Postings list containing common items of 'kent' and 'disguised'
20 67
```

Limitation and Future Scope:

The **limitations** of using skip pointers for faster posting list intersection include the increased memory usage due to the need to store the additional pointers. Additionally, the performance gain may not be significant if the posting lists are small or have few common elements.

The **future scope** for improving the efficiency of posting list intersection using skip pointers may include the use of more advanced data structures, such as compressed skip pointers or multi-level indexing. Another approach could be to use machine learning techniques to predict which posting lists are likely to have a high degree of overlap, and focus on intersecting those lists first. Additionally, parallelization and distributed computing can be utilized to speed up the intersection process.

Conclusion:

In conclusion, skip pointers are a useful technique for improving the efficiency of posting list intersection in information retrieval systems. By adding additional pointers to the posting lists, skip pointers allow the system to skip over non-overlapping sections and quickly identify and retrieve the documents that contain the common terms. However, the use of skip pointers also increases memory usage, which can be a limitation. To overcome this limitation, advanced data structures like compressed skip pointers or multi-level indexing, machine learning techniques and parallelization can be used to further improve the efficiency of the intersection process.