Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

# Курсовой проект по курсу «Дискретный анализ»
# Текстовый Поиск

Студент: Кузьмичев А. Н.
Преподаватель: Ридли А.Н.
Группа: М8О-306Б-18
Дата:
Оценка:
Подпись:

Москва, 2021

# Курсовой проект

**Задание:** Необходимо реализовать программу, выполняющую поиск слов в статьях Википедии. Программа должна работать в двух режимах:

• Для обработки файла в котором будет происходит поиск.

• Для поиска в обработанном файле.

Доступны логические операции &(логическое и), |(логическое или), ~(логическое не) для связывания слов в запросах. Вывод программы на запрос может существовать в двух вариантах, либо программа выводит количество статей, попадающих под запрос, либо количество и название всех статей, в которых запрос был найден.

# 1. Теоретическая часть: описание принципов работы программы, алгоритмы

Реализацию программы можно разделить на две части.

1) Реализация структуры данных инвертированный индекс для данных Википедии. Кодировка данных в simple9-кодировку и сохранение данной структуры в файл.
2) Реализация парсера для обработки выражения запроса. Обеспечение корректной работы с логическими выражениями в процессе поиска по данным.

Инвертированный индекс – структура данных, где для каждого слова множества документов в соответствующем списке перечислены все документы, в которых это слово содержится. Инвертированный индекс часто используется в текстовом поиске. В данной программе необходим инвертированный индекс, содержащий только список документов для каждого слова. Стоит отметить, что также существует реализация, в которой помимо номера документа хранится позиция слова в документе. Также используются дополнительные метрики, которые применяются в дальнейшем при ранжировании. К преимуществам инвертированного индекса можно отнести то, что благодаря ему возможно быстро найти документы, в которых встречаются искомые слова.

Simple9 – метод, кодирующий целочисленные значения. Для сжатия данных используются машинные слова из 32 бит, 4 бита под управление и 28 под хранимые данные. Обычные числа хранятся в 32 битном слове, но множество битов не используются для хранения числа. Например число 7 использует только 3 первых бита, остальные 29 битов остаются неиспользуемыми, на этом и основан метод Simple9. То есть 9 чисел которые используют 3 бита могут хранится в одном слове simple9, один бит останется неиспользуемым. И так далее для всех чисел до 2 28.

Алгоритм сортировочной станции – способ разбора математических выражений, представленных в инфиксной нотации. Может быть использован для получения вывода в виде обратной польской нотации или в виде абстрактного синтаксического дерева. Алгоритм изобретен Эдсгером Дейкстрой.  Название «алгоритм сортировочной станции» было придумано им же, поскольку напоминает действие железнодорожной сортировочной станции. Благодаря этому методу можно перевести из инфиксной записи в постфиксную, а из постфиксной легко посчитать выражение. Этот алгоритм можно модифицировать и сразу считать выражение по мере разбора.

# 2.Имплементация

Как уже упоминалось выше, реализация программы разделена на 2 части.

1) **Индексирование**

Программа производит парсинг входных данных, в данном случае, Википедии. Каждой статье, до того, как ее текст будет обработан, присваивается id, уникальный идентификатор. Затем, в процессе обработки каждого слова статьи, программа добавляет id в инвертированный индекс с этим словом. Если в одном слове накопилось 28 номеров, программа кодирует эти номера методом Simple9. После завершения процесса парсинга программа сбрасывает таблицу номеров - названий статей и инвертированный индекс в файл.

Запуск программы для индексации

*./prog index --input <input file> --output <index file>*

2) **Поиск слов**

Сначала происходит выгрузка данных из индексированного файла в инвертированный индекс. Процесс поиска слов – это обработка файла с запросами. Для обработки поискового выражения используется модификация алгоритма сортировочной станции Дейкстры. При обработке выражения программа берет из этого выражения слова, извлекает номера статей, в которых встречаются эти слова, с помощью инвертированного

индекса,  и вычисляет результат выражения, используя такие базовые операции теории множеств,  как пересечение, объединение и отрицание множеств. Результат будет записан в файл.

Так же в программе реализована функция *--full-output.* Если она указана - на каждый запрос выводится отдельная строка, с количеством документов подпадающих под запрос, а затем названия всех документов подпадающих под запрос по одному названию в строке. На каждый запрос в отдельной строке выводится количество документов подпадающих под запрос.

Запуск программы для поиска слов

*./prog search --index <index file>  --input <input file> --output <output file> [--full-output]*

# 3. Примеры работы программ, форматы файлов

**Формат файлов для индексации:**

*<doc id="1951221" url=" https://en.wikipedia.org/wiki?curid=1951221">*

*<Text>*

*</doc>*

**Формат файлов для поиска:**

*<word 1>*

*<word 1> & <word 2>*

*~<word 1>*

*<word 1> | <word 2>*

*~(<word 1> & <word 2> & <word 3>) | (<word 4> & (<word 5> | ~<word 6>))*

*Примеры работы программы:*

*<doc id="19001" url = https://en.wikipedia.org/wiki?curid=19001 title = “Microsoft”>*

*Microsoft*

*Microsoft Corporation (/ˈmaɪkrəsɒft/, /-kroʊ-/) is an American multinational technology company with headquarters in Redmond, Washington. It develops, manufactures, licenses, supports, and sells computer software, consumer electronics, personal computers, and related services. Its best known software products are the Microsoft Windows line of operating systems, the Microsoft Office suite, and the Internet Explorer and Edge web browsers. Its flagship hardware products are the Xbox video game consoles and the Microsoft Surface lineup of touchscreen personal computers. Microsoft ranked No. 21 in the 2020 Fortune 500 rankings of the largest United States corporations by total revenue; it was the world's largest software maker by revenue as of 2016. It is considered one of the Big Five companies in the U.S. information technology industry, along with Google, Apple, Amazon, and Facebook.*

*</doc>*

*<doc id="1951221" url = https://en.wikipedia.org/wiki?curid=1951221 title = “Secretly Canadian”>*

*Secretly Canadian*

*Secretly Canadian is an American independent record label based in Bloomington, Indiana with offices in New York, Los Angeles, Chicago, Austin, London, Paris, Amsterdam, and Berlin. Secretly Canadian is a label included in Secretly Group,*

*which also includes Dead Oceans and Jagjaguwar. Secretly Group includes the three record labels as well as a music publisher known as Secretly Publishing, representing artists, writers, film makers, producers, and comedians.*

*</doc>*

**Запросы:**

*apple*

*music*

*~record*

*apple & record*

**Вывод программы:**

***Без функции –full-output***

*1*

*1*

*1*

*0*

***С функцией –full-output***

*1*

*Microsoft*

*1*

*Secretly Canadian*

*1*

*Microsoft*

*0*

# 4. Выводы

Выполняя курсовой проект, я познакомился с основами информационного поиска, его базовыми алгоритмами и принципами. Например, мне пришлось досконально изучить такую структуру данных, как инвертированный индекс. Инвертированный индекс — это самая популярная структура данных, которая используется в информационном поиске. Она находится под капотом практически каждого поисковика. У нее существуют различные модификации. Например, обычно в поисковых системах после построения с помощью инвертированного индекса списка документов, содержащих слова из запроса, идет ранжирование документов из списка. Поэтому, для дальнейшего ранжирования, в списке вхождений слова в документы, помимо id документов, обычно также указываются различные факторы.  К примеру, TF-IDF ( TF — term frequency, IDF — inverse document frequency), BM25 (best match) отображают степень релевантности документа запросу одним числом. Чем выше значение метрик, тем более релевантен документ. Так же принимается во внимание факт того, попало ли в заголовок слово, или нет.

Моя версия текстового поиска весьма примитивна по сравнению с использующимися в продуктиве. Тем не менее, эта курсовая работа заставила меня углубится в изучение текстового поиска. Так же мне пришлось вспомнить изученный на первом курсе алгоритм сортировочной станции, так как было разумнее всего реализовать через него операции над множествами для составных запросов.

# 5.Листинг кода

***main.cpp***

```
#include "TInvertedIndex.h"
#include <ctime>
#include <cstring>


int main(int argc, char **argv) {
   //unsigned int start_time =  clock();

   std::locale::global(std::locale("en_US.UTF-8"));


   if (argc < 2) {
      std::cout << "Arguments error: missing name of action\n";
      return 0;
   }

   if (strcmp(argv[1], "index") == 0) {

      if (argc < 6) {
         std::cout << "Arguments error: wrong input or output arguments\n";
         return 0;
```

```
}

std::wifstream input_file;
std::wofstream output_file;

for (int i = 2; i < argc; i += 2) {
    if (strcmp(argv[i], "--input") == 0) {
        if (argc - 1 == i) {
            std::cout << "Argument error\n";
            return 1;
        }
        std::string input_file_name(argv[i + 1]);
        input_file.open(input_file_name);
        if (!input_file.is_open()) {
            std::cout << "File error: file doesn't exist\n";
            return 1;
        }
    } else {
        if (argc - 1 == i) {
            std::cout << "Argument error\n";
            return 1;
        }
        std::string output_file_name(argv[i + 1]);
        output_file.open(output_file_name);
```

```
        if (!output_file.is_open()) {
            std::cout << "File error: output file error\n";
        }
    }
}

TInvertedIndex iv;
iv.Read(input_file);
iv.Save(output_file);

} else if (strcmp(argv[1], "search") == 0) {

if (argc < 8) {
    std::cout << "Arguments error: wrong input or output arguments\n";
    return 0;
}

TInvertedIndex iv;

std::wifstream index_file;
std::wifstream input_file;
std::wofstream output_file;
```

```
for (int i = 2; i < argc; ++i) {
    if (strcmp(argv[i], "--index") == 0) {
        if (argc - 1 == i) {
            std::cout << "Argument error\n";
            return 1;
        }
        ++i;
        std::string index_file_name(argv[i]);
        index_file.open(index_file_name);
        if (!index_file.is_open()) {
            std::cout << "File error: file doesn't exist\n";
        }
    }
    else if (strcmp(argv[i], "--input") == 0) {
        if (argc - 1 == i) {
            std::cout << "Argument error\n";
            return 1;
        }
        ++i;
        std::string input_file_name(argv[i]);
        input_file.open(input_file_name);
        if (!input_file.is_open()) {
            std::cout << "File error: file doesn't exist\n";
```

```
        }
      }
      else if (strcmp(argv[i], "--output") == 0) {
        if (argc - 1 == i) {
          std::cout << "Argument error\n";
          return 1;
        }
        ++i;
        std::string output_file_name(argv[i]);
        output_file.open(output_file_name);
        if (!output_file.is_open()) {
          std::cout << "File error: output file error\n";
        }
      }
      else if (strcmp(argv[i], "--full-output") == 0) {
        iv.full_output = true;
      }
    }

    iv.Load(index_file);
    iv.ReadQueries(input_file, output_file);


  } else {
```

```
        std::cout << "Arguments error: wrong action\n";
        return 0;
    }


//    unsigned int end_time = clock();
//    unsigned int search_time = end_time - start_time;
//    std::cout << search_time << "\n";
    return 0;
}
```

***TSimple9.h***

```
#ifndef KP_SIMPLE9_H
#define KP_SIMPLE9_H


#include <iostream>
#include <vector>

#include <unordered_map>
#include <algorithm>
#include <fstream>
#include <string>
```

```
#include <stack>
#include <sstream>

#include <assert.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <limits.h>



#define SELECTOR_MASK 0x0000000F //15

#define SELECTOR_BITS 4

#define NSELECTORS 9

static const struct {
   uint32_t nitems;
   uint32_t nbits;
   uint32_t nwaste;
} selectors[NSELECTORS] = {
```

```
    {28, 1,  0},
    {14, 2,  0},
    {9,  3,  1},
    {7,  4,  0},
    {5,  5,  3},
    {4,  7,  0},
    {3,  9,  1},
    {2,  14, 0},
    {1,  28, 0},
};


void TSimple9_encode(std::vector<uint32_t> &array, std::vector<uint32_t>
&encodeArray);

void TSimple9_decode(std::vector<uint32_t> &vec, std::vector<uint32_t > &ans);


#endif //KP_SIMPLE9_H
```

***TSimple9.cpp***

```
#include "TSimple9.h"
```

```
void TSimple9_encode(std::vector<uint32_t> &array, std::vector<uint32_t>
&encodeArray) {

    uint32_t index;
    uint32_t selector;
    uint32_t data;
    uint32_t shift;

    size_t nitems;
    size_t i;
    size_t arraySize = array.size();



    index = 0;



    while (index < arraySize) {
        for (selector = 0; selector < NSELECTORS; selector++) { // choose selector
            data = selector; // control bits
            shift = SELECTOR_BITS; // start 4 bits
            nitems = 0; // count of elements in one uint_32

            for (i = index; i < arraySize; ++i) { // filling selector

                if (nitems == selectors[selector].nitems) // selector is full
```

```
        break;

    // element in array more then max in this selector
    if (array[i] > (1UL << selectors[selector].nbits) - 1)
        break;

    // add element in data
    // {element 1001} or {element element 1000}
    data |= (array[i] << shift);

    shift += selectors[selector].nbits;

    nitems++;
}

// selector is full or elements are over
if (nitems == selectors[selector].nitems || index + nitems == arraySize) {

    encodeArray.push_back(data);
    index += nitems;

    break;
}
```

```
        } /* End for selector ... */

    } /* End while index < n */
}

void TSimple9_decode(std::vector<uint32_t> &vec, std::vector<uint32_t > &ans) {

    uint32_t lastElement = 0;
    uint32_t element;

    for (auto selector : vec) { //take selector
        uint32_t typeSelector = selector & SELECTOR_MASK;

        uint32_t  countOfElements = selectors[typeSelector].nitems; // count of
elements in one selector

        uint32_t shift = selectors[typeSelector].nbits;

        uint32_t mask = (1u << selectors[typeSelector].nbits) - 1;

        selector = selector >> SELECTOR_BITS; // skip control bits

        for (int i = 0; i < countOfElements; ++i) {
            element = (selector & mask);
            if (element) {
```

```
            if (lastElement != 0) {
                // remove deltas
                lastElement += element;
                ans.push_back(lastElement);
            } else {
                lastElement = element;
                ans.push_back(element);
            }
        }
        selector = selector >> shift;
    }
  }

}
```

***TInvertedIndex.h***

```
#ifndef KP_INVERTEDINDEX_H
#define KP_INVERTEDINDEX_H

#include "TEnc.h"

class TInvertedIndex {
public:
```

```
    TInvertedIndex();

    void Read(std::wifstream &in);
    void ReadArticle(std::wifstream &in, uint32_t &numOfArticle);
    void Load(std::wifstream &in);
    void Save(std::wofstream &out);


    ////////////////
    ////for queries
    ////////////////
    void ReadQueries(std::wifstream &in, std::wofstream &out);
    void Query(std::wstring &str, std::wofstream &out);

    int Priority(wchar_t &op);

    void Evaluate(std::vector<uint32_t> &val1, wchar_t &op, std::vector<uint32_t>
&val2, std::vector<uint32_t> &result);
    void Evaluate(wchar_t &op, std::vector<uint32_t> &val1, std::vector<uint32_t>
&result);

    void QueryIntersection(std::vector<uint32_t> &lhs, std::vector<uint32_t> &rhs,
std::vector<uint32_t> &result);
    void QueryUnion(std::vector<uint32_t> &lhs, std::vector<uint32_t> &rhs,
std::vector<uint32_t> &result);
```

```
    void QueryDifference(std::vector<uint32_t> &lhs, std::vector<uint32_t> &rhs,
std::vector<uint32_t> &result);

    bool full_output;
private:
    // <word> <num of article>
    std::unordered_map<std::wstring, TEnc> words;
    // <num of article> <name of article>
    std::vector<std::wstring> names;
    std::vector<uint32_t > nums;


};


#endif //KP_INVERTEDINDEX_H
```

***TInvertedIndex.cpp***

```
#include "TInvertedIndex.h"

//#include <sys/time.h>

TInvertedIndex::TInvertedIndex() : full_output(false) {}
```

```
void TInvertedIndex::Read(std::wifstream &in) {
   //find title

   std::wstring str;
   std::wstring nameOfDoc;
   uint32_t numOfArticle = 1;

   while (getline(in, str)) {
      if (!str.empty()) {
         //take name of doc
         size_t pos = str.find(L"title");
         nameOfDoc = str.substr(pos + 7, str.size() - pos - 7 - 2);

         //read doc's inner
         ReadArticle(in, numOfArticle);
         numOfArticle++;

         //push name of doc
         names.push_back(nameOfDoc);
      }
   }

}
```

```
void TInvertedIndex::ReadArticle(std::wifstream &in, uint32_t &numOfArticle) {

    std::wstring str;
    std::wstring word;
    while (in >> str) {

        if (str == L"</doc>") {
            break;
        }

        //convert the word into a normal form
        for (auto &c : str) {
            if (iswalnum(c)) {
                word += static_cast<wchar_t >(towlower(c));
            }
            else if (!word.empty()) {
                words[word].Push(numOfArticle);
                word.clear();
            }

        }

        if (!word.empty())
            words[word].Push(numOfArticle);
```

```
        word.clear();
    }
}

void TInvertedIndex::Save(std::wofstream &out) {

    for (auto &i : names) {
        out << i << L"\n";
    }
    out << L"_\n";
    for (auto &i : words) {
        out << i.first << L"\n";

        i.second.PushDelay();
        i.second.Save(out);
    }
    out << L"_";
}

void TInvertedIndex::Load(std::wifstream &in) {
    uint32_t numOfArticle = 1;
    std::wstring str;
```

```
    while (true) {
        getline(in, str);
        if (str == L"_")
            break;
        names.push_back(str);
        nums.push_back(numOfArticle);
        numOfArticle++;
    }
    while (true) {
        in >> str;
        if (str == L"_")
            break;
        words[str].Load(in);
    }

}

////////////// for queries


int TInvertedIndex::Priority(wchar_t &op) {
    if (op == L'~') return 3;
    if (op == L'&') return 2;
    if (op == L'|') return 1;
```

```
    return 0;
}


void
TInvertedIndex::QueryIntersection(std::vector<uint32_t> &lhs,
std::vector<uint32_t> &rhs,
                                  std::vector<uint32_t> &result) {

    std::set_intersection(lhs.begin(), lhs.end(),
                          rhs.begin(), rhs.end(),
                          std::back_inserter(result));

}

void TInvertedIndex::QueryUnion(std::vector<uint32_t> &lhs,
std::vector<uint32_t> &rhs, std::vector<uint32_t> &result) {

    std::set_union(lhs.begin(), lhs.end(),
                   rhs.begin(), rhs.end(),
                   std::back_inserter(result));

}

void TInvertedIndex::QueryDifference(std::vector<uint32_t> &lhs,
```

```
                        std::vector<uint32_t> &rhs, std::vector<uint32_t> &result) {
  std::set_difference(lhs.begin(), lhs.end(),
                      rhs.begin(), rhs.end(),
                      std::back_inserter(result));
}

void
TInvertedIndex::Evaluate(std::vector<uint32_t> &val1, wchar_t &op,
std::vector<uint32_t> &val2,
                       std::vector<uint32_t> &result) {


  switch (op) {
    case L'&':
      QueryIntersection(val1, val2, result);
      break;
    case L'|':
      QueryUnion(val1, val2, result);
      break;
    default:
      break;
  }
}
```

```
void TInvertedIndex::Evaluate(wchar_t &op, std::vector<uint32_t> &val1,
std::vector<uint32_t> &result) {
   QueryDifference(nums, val1, result);
}

void TInvertedIndex::ReadQueries(std::wifstream &in, std::wofstream &out) {
   std::wstring str;
   while (getline(in, str))
      Query(str, out);
}

void TInvertedIndex::Query(std::wstring &str, std::wofstream &out) {

   /*struct timeval  tv;
   gettimeofday(&tv, NULL);
   double time_begin = ((double)tv.tv_sec) * 1000 +
                  ((double)tv.tv_usec) / 1000;
*/
   std::vector<uint32_t> emptyVec;

   //parse and evaluate expression
   std::stack<wchar_t> operators;
   std::stack<std::vector<uint32_t>> values;

   for (int i = 0; i < str.size(); ++i) {
```

```
//skip whitespaces
if (str[i] == L' ')
    continue;

if (str[i] == L'(') {
    operators.push(L'(');
} else if (iswalnum(str[i])) { // if it's word
    std::wstring word;
    while (i < str.size() && iswalnum(str[i])) {
        word += static_cast<wchar_t >(towlower(str[i]));
        i++;
    }
    i--;
    if (words.find(word) != words.end()) {
        std::vector<uint32_t> articles;
        words[word].Decode(articles);
        values.push(articles);
    } else {
        values.push(emptyVec);
    }
} else if (str[i] == L')') {
    while (!operators.empty() && operators.top() != L'(') {
        std::vector<uint32_t> tmp;
```

```
            wchar_t op = operators.top();
            operators.pop();

            if (op == L'~') {

                Evaluate(op, values.top(), tmp);
                tmp.swap(values.top());

            } else {

                std::vector<uint32_t> val1 = values.top();
                values.pop();

                Evaluate(val1, op, values.top(), tmp);
                tmp.swap(values.top());
            }
        }
        operators.pop();

    } else { //else it operator
        wchar_t rightOp = str[i];
        while (!operators.empty() && Priority(operators.top()) >= Priority(rightOp))
{
            std::vector<uint32_t> tmp;
            wchar_t op = operators.top();
```

```
            operators.pop();

            if (op == L'~') {
                Evaluate(op, values.top(), tmp);
                tmp.swap(values.top());
            } else {

                std::vector<uint32_t> val1 = values.top();
                values.pop();

                Evaluate(val1, op, values.top(), tmp);
                tmp.swap(values.top());
            }
        }
        operators.push(rightOp);
    }
}

while (!operators.empty()) {
    std::vector<uint32_t> tmp;
    wchar_t op = operators.top();
    operators.pop();

    if (op == L'~') {
```

```
        Evaluate(op, values.top(), tmp);
        tmp.swap(values.top());
    } else {
        std::vector<uint32_t> val1 = values.top();
        values.pop();

        Evaluate(val1, op, values.top(), tmp);
        tmp.swap(values.top());
    }
}

//write answer
if (!values.empty()) {
    std::vector<uint32_t> ans = values.top();
    if (!ans.empty()) {
        out << ans.size() << L"\n";
        if (full_output)
            for (auto &i : ans) {
                out << names[i - 1] << L"\n";
            }
    } else {
        out << 0 << L"\n";
    }
} else {
```

```
        out << 0 << L"\n";
    }

    /*gettimeofday(&tv, NULL);
    double time_end = ((double)tv.tv_sec) * 1000 +
                ((double)tv.tv_usec) / 1000 ;
    double total_time_ms = time_end - time_begin;
    printf("TOTAL TIME (ms) = %f\n", total_time_ms);*/

}
```

***TEnc.h***

```
#ifndef KP_ENC_H
#define KP_ENC_H

#include "TSimple9.h"

class TEnc {
public:
    TEnc() = default;


    void Push(uint32_t &numOfText);
```

```
    //encode all delays nums
    void PushDelay();

    void Decode(std::vector<uint32_t> &ans);

    void Save(std::wofstream &out);
    void Load(std::wifstream &in);


    std::vector<uint32_t> encoded;
    std::vector<uint32_t> delay;
    uint32_t lastNum = 0;
};


#endif //KP_ENC_H
```

***TEnc.cpp***

```
#include <stdint-gcc.h>
#include "TEnc.h"


void TEnc::Push(uint32_t &numOfText) {
```

```
    if (lastNum != 0) {
        if (lastNum != numOfText) {
            delay.push_back(numOfText - lastNum);
            lastNum = numOfText;
        } else {
            return;
        }
    } else {
        delay.push_back(numOfText);
        lastNum = numOfText;
    }


    if (delay.size() == 28) {
        TSimple9_encode(delay, encoded);
        delay.clear();
    }
}

void TEnc::PushDelay() {
    if (!delay.empty()) {
        TSimple9_encode(delay, encoded);
        delay.clear();
    }
```

```
}

void TEnc::Decode(std::vector<uint32_t> &ans) {
   if (!delay.empty()) {
      TSimple9_encode(delay, encoded);
      delay.clear();
   }
   TSimple9_decode(encoded, ans);

}

void TEnc::Save(std::wofstream &out) {

   for (auto i : this->encoded) {
      out << i << L"\n";
   }
   out << L"0\n";
}

void TEnc::Load(std::wifstream &in) {
   uint32_t num;
   while (true) {
      in >> num;
      if(num == 0)
```

```
            break;
        encoded.push_back(num);
    }
}
```