МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

ЗВІТ

з лабораторної роботи №1
«Реалізація моделей подання документів»

Виконав:
студент 4-го курсу, групи КП-91, спеціальності 121 – Інженерія програмного забезпечення
Власюк Сергій Петрович

Перевірив:
ас. каф. ПЗКС
Юсин Яків Олексійович

Київ 2023

Постановка задачі за варіантом

Реалізувати теоретико-множинну (стандартну булеву) та алгебраїчну (векторно-просторову) модель подання документів. Робота з розробленим програмним забезпеченням повинне задовільнятися наступними вимогами:

- Режим з використанням стандартної булевої моделі подання
  - Етап 1. Введення множини індексних термів
  - Етап 2. Введення колекції документів
  - Етап 3. Виконання пошукових запитів
- Режим з використанням векторно-просторової моделі
  - Етап 1. Введення колекції документів
  - Етап 2. Виконання пошукових запитів

Для реалізації векторно-просторової моделі використаємо міру TF-IDF відповідно до варіанту №4

4	$f_{t,d}$	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t}\right)$

Рис.1 tf-idf міра

1)Реалізація стандартної булевої моделі

```
const termInput = document.getElementById('term');
const documentInput = document.getElementById('document');
const searchInput = document.getElementById('search');
const termForm = document.getElementById('term-form');
const documentForm = document.getElementById('document-form');
const searchForm = document.getElementById('search-form');
const termResult = document.getElementById('term-result');
const documentResult = document.getElementById('document-result');
const searchResult = document.getElementById('search-result');
const searchqueryResult = document.getElementById('search-query');
const termSection = document.getElementById('terms-section');
const documentSection = document.getElementById('documents-section');
const searchSection = document.getElementById('search-section');
const termResultSection = document.getElementById('terms-result-section');
```

```
const nextStageButton = document.getElementById('document-next-stage-button');
const documentresultSection = document.getElementById('documention-result-section');
const searchResultSection = document.getElementById('search-result-section');

documentSection.style.display = 'none';
searchSection.style.display = 'none';
documentresultSection.style.display = 'none';
// ____________________________________________________
let terms;
let documents = [];
let documentsObject;
let currentDocId = 0;

const onTermFormSubmit = event => {
  event.preventDefault();
  const text = termInput.value;
  if (!text) {
        alert('Enter at least one term');
        return;
  }
  terms = [
        ...new Set(
        text
        .toLowerCase()
        .split(', ')
        .map(line => line.trim()),
        ),
  ];

  const termsToDisplay = terms.map((term, i) => `term ${i + 1}: "${term}"`).join('<br>');
  termSection.style.display = 'none';
  termResultSection.style.display = 'block';
  document.getElementById('terms-result').innerHTML = termsToDisplay;
  documentSection.style.display = 'block';
};
```

```
const onNextStageClicked = () => {
  if (!documents?.length || documents?.length < 1) {
        alert('Enter at least one document');
        return;
  }
  documentSection.style.display = 'none';
  searchSection.style.display = 'block';
};

const onDocumentFormSubmit = event => {
  event.preventDefault();
  const text = documentInput.value;
  if (!text) {
        alert('Enter the contents of the document');
        return;
  }
  documentresultSection.style.display = 'block';
  documents.push(text);
  document.getElementById('documention-result').innerHTML = documents
        .map((doc, i) => `document ${i + 1}: "${doc}"`)
        .join('<br>');
  documentsObject = createDocumentObject(documents, terms);
  documentForm.reset();
};

const onSearchFormSubmit = event => {
  event.preventDefault();
  const text = searchInput.value;
  if (!text) {
        alert(''); //search emty
        return;
  }
  if (!isDNF(text)) {
        alert(''); //not dnf form
        return;
  }
```

```
  const searchRes = booleanModel(text, documentsObject);
  if (!searchRes) {
        return;
  }
  searchResultSection.style.display = 'block';
  searchqueryResult.innerHTML = text;
  searchResult.innerHTML = searchRes?.length
        ? `Documents matching the search results: ${searchRes}`
        : 'Documents not found';
  searchForm.reset();
};

termForm.addEventListener('submit', onTermFormSubmit);
nextStageButton.addEventListener('click', onNextStageClicked);
documentForm.addEventListener('submit', onDocumentFormSubmit);
searchForm.addEventListener('submit', onSearchFormSubmit);

// ____________________________________________________________
function isDNF(query) {
  if (
        query.includes(') AND') ||
        query.includes('AND (') ||
        query.includes('NOT (') ||
        query.includes('NOT NOT')
  ) {
        return false;
  }
  return true;
}

const getTerms = query =>
  query
        .split(/(\(|\)|\s+OR\s+|\s+AND\s+|\s+NOT\s+)/)
        .map(term => term.trim())
        .filter(term => !!term);
```

```
const getSubTerms = (terms = []) => {
  let subterms;
  if (terms.includes('(')) {
            subterms = [[]];
            let subtermIndex = 0;
            let skipIterationIdx;
            terms.forEach((term, i) => {
            if (i !== skipIterationIdx) {
            if (term === ')') {
            subtermIndex += 1;
            subterms[subtermIndex] = [];
            if (terms?.[i + 1]) {
            subterms[subtermIndex].push(terms?.[i + 1]);
            subtermIndex += 1;
            subterms[subtermIndex] = [];
            skipIterationIdx = i + 1;
            }
            }
            if (term === '(') {
            if (terms?.[i - 1] && terms?.[i - 2] !== ')') {
            subterms[subtermIndex].pop();
            subtermIndex += 1;
            subterms[subtermIndex] = [terms?.[i - 1]];
            }
            subtermIndex += 1;
            subterms[subtermIndex] = [];
            }
            if (term !== ')' && term !== '(') {
            subterms[subtermIndex].push(term);
            }
            } else {
            skipIterationIdx = undefined;
            }
            });
  }
  return subterms?.filter(el => el.length > 0);
```

```
};

const searchBySubterms = (terms, docs) => {
  let result = [];
  for (let i = 0; i < terms.length; i++) {
            const term = terms[i];

            if (term === 'AND' || term === 'OR' || term === 'NOT' || term === '(' || term === ')') {
            continue;
            } else if (term.startsWith('NOT ')) {
            const notTerm = term.substring(4);
            const notResult = [];

            for (const doc in docs) {
            if (!docs.hasOwnProperty(doc)) {
            continue;
            }
            if (!docs[doc].includes(notTerm)) {
            notResult.push(doc);
            }
            }

            if (notResult.length > 0) {
            if (result.length === 0) {
            result = notResult;
            } else if (terms[i - 1] === 'OR') {
            result = [...new Set([...result, ...notResult])];
            } else if (terms[i - 1] === 'AND') {
            result = result.filter(doc => notResult.includes(doc));
            }
            }
            } else {
            const termResult = [];

            for (const doc in docs) {
            if (!docs.hasOwnProperty(doc)) {
```

```
            continue;
            }

            if (docs[doc].includes(term)) {
            termResult.push(doc);
            }
            }
            if (termResult.length > 0) {
            if (result.length === 0) {
            result = termResult;
            } else if (terms[i - 1] === 'OR') {
            result = [...new Set([...result, ...termResult])];
            } else if (terms[i - 1] === 'AND') {
            result = result.filter(doc => termResult.includes(doc));
            }
            }
            }
  }
  return result;
};

function booleanModel(query, docs) {
  const newTerms = getTerms(query);
  let shouldQuit = false;
  newTerms.forEach(newTerm => {
            if (
            newTerm !== 'AND' &&
            newTerm !== 'OR' &&
            newTerm !== 'NOT' &&
            newTerm !== '(' &&
            newTerm !== ')'
            ) {
            if (newTerm.startsWith('NOT ')) {
            newTerm = newTerm.substring(4);
            }
            if (!terms.includes(newTerm)) {
```

```
        shouldQuit = true;
        console.log(newTerm);
        alert('Unknown term in query');
        }
        }
});
if (shouldQuit) {
        return false;
}
const subterms = getSubTerms(newTerms);

if (subterms?.length > 0) {
        const subresults = subterms.map((sterms, i) => {
        if (i % 2 === 0) {
        return searchBySubterms(sterms, docs);
        }
        return sterms[0];
        });
        let result = [];
        for (let i = 1; i < subresults.length - 1; i += 2) {
        if (result.length === 0) {
        if (i === 1) {
        if (subresults[i] === 'OR') {
        result = [...subresults[i - 1], ...subresults[i + 1]];
        } else if (subresults[i] === 'AND') {
        result = subresults[i - 1].filter(doc => subresults[i + 1].includes(doc));
        }
        } else {
        if (subresults[i] === 'OR') {
        result = [...result, ...subresults[i + 1]];
        } else if (subresults[i] === 'AND') {
        result = result.filter(doc => subresults[i + 1].includes(doc));
        }
        }
        } else if (result.length > 0) {
        if (subresults[i] === 'OR') {
```

```
            result = [...result, ...subresults[i + 1]];
            } else if (subresults[i] === 'AND') {
            result = result.filter(doc => subresults[i + 1].includes(doc));
            }
            }
            }
            return [...new Set(result)];
  }

  return searchBySubterms(newTerms, docs);
}
function getWordsFromString(str) {
  const strippedStr = str.replace(/[^\w\s]/gi, '').toLowerCase();
  const words = strippedStr.split(/\s+/);

  return words;
}

function createDocumentObject(documents, terms) {
  const docs = {};
  for (let i = 0; i < documents.length; i++) {
            const doc = getWordsFromString(documents[i]);
            const docTerms = [];
            for (let j = 0; j < terms.length; j++) {
            const term = terms[j];
            if (doc.includes(term)) {
            docTerms.push(term);
            }
            }
            docs[i + 1] = docTerms;
  }
  return docs;
}
```

2)Реалізація векторно-просторової моделі

```
const documentInput = document.getElementById('document');
```

```
const searchInput = document.getElementById('search');
const documentForm = document.getElementById('document-form');
const searchForm = document.getElementById('search-form');
const documentResult = document.getElementById('document-result');
const searchResult = document.getElementById('search-result');
const searchqueryResult = document.getElementById('search-query');
const documentSection = document.getElementById('documents-section');
const searchSection = document.getElementById('search-section');
const nextStageButton = document.getElementById('document-next-stage-button');
const documentresultSection = document.getElementById('documention-result-section');
const searchResultSection = document.getElementById('search-result-section');

documentSection.style.display = 'block';
searchSection.style.display = 'none';
// ____________________________________________________
let documents = [];
let currentDocId = 0;
let doc_vectors = [];
let idf;
const threshold = 0.15;

const onNextStageClicked = () => {
  if (!documents?.length || documents?.length < 1) {
            alert('No documents!');
            return;
  }
  documentSection.style.display = 'none';
  searchSection.style.display = 'block';

  idf = calculate_idf(documents);
  doc_vectors = calculate_tf_idf(documents, idf);
};

const onDocumentFormSubmit = event => {
  event.preventDefault();
  const text = documentInput.value;
```

```
  if (!text) {
        alert('Document is empty!');
        return;
  }
  documentresultSection.style.display = 'block';
  documents.push(text);
  document.getElementById('documention-result').innerHTML = documents
        .map((doc, i) => `document ${i + 1}: "${doc}"`)
        .join('<br>');
  documentForm.reset();
};

const onSearchFormSubmit = event => {
  event.preventDefault();
  const query = searchInput.value;
  if (!query) {
        alert('Enter searh query');
        return;
  }
  // CЂP°C…CґC” tfidf
  const query_vector = calculate_tf(query, idf);

  const results = [];
  for (let i = 0; i < documents.length; i++) {
        const similarity = cosine_similarity(query_vector, doc_vectors[i]);
        if (similarity > threshold) {
        results.push([i, similarity]);
        }
  }

  results.sort((a, b) => b[1] - a[1]);

  intervals = '';

  for (const [index, similarity] of results) {
        intervals += `similarity document ${index + 1}: ${similarity.toFixed(3)}<br>`;
```

```
  }

  searchResultSection.style.display = 'block';
  searchqueryResult.innerHTML = query;
  searchResult.innerHTML = results.length ? intervals : 'Documents';
  searchForm.reset();
};

nextStageButton.addEventListener('click', onNextStageClicked);
documentForm.addEventListener('submit', onDocumentFormSubmit);
searchForm.addEventListener('submit', onSearchFormSubmit);

// ____________________________________________________________
function preprocess(text) {
  console.log(text.toLowerCase().match(/\w+/g));
  return text
            .replace(/[^\w\s]/gi, '')
            .toLowerCase()
            .split(/\s+/);
}

// inverse document frequency
function calculate_idf(documents) {
  const maxnt = {};
  const nt = {};
  const N = documents.length;
  let maxnt_num = 0;
  for (const document of documents) {
            const terms = preprocess(document);

            for (const term of terms) {
            maxnt[term] = (maxnt[term] || 0) + 1;
            }
  }
  console.log(maxnt)
```

```
  for (const document of documents) {
        const terms = preprocess(document);
        const newSet = new Set(terms); //
        const terms_uniq = Array.from(newSet);
        for (const term of terms_uniq) {
        nt[term] = (nt[term] || 0) + 1;
        }
  }

  // Idf PїPs C„PsCЂPjCѓP»C–
  const updatedIdf = {};

  for (const term in maxnt) {
        if (N - maxnt[term] === 0 || maxnt[term] === 0) {
        updatedIdf[term] = 0;
        } else {
        updatedIdf[term] = Math.log(nt[term] / (1 + maxnt[term]));
        }
  }

  return updatedIdf;
}
// CЂP°C…CѓC” tfїdf
function calculate_tf(query, idf) {
  const tf = {};
  const terms = preprocess(query);

  for (const term of terms) {
        tf[term] = (tf[term] || 0) + 1;
  }

  const vector = [];

  for (const term in idf) {
        const tf_score = (tf[term] || 0);
        const tf_idf_score = tf_score * idf[term];
```

```
            vector.push(tf_idf_score);
  }
  return vector;
}

function calculate_tf_idf(documents, idf) {
  const tf_idf = [];

  for (const document of documents) {
            // term frequency
            // CЂP°C…CґC” tfidf
            const vector = calculate_tf(document, idf);
            tf_idf.push(vector);
  }

  return tf_idf;
}

function cosine_similarity(a, b) {
  let sum = 0;
  let normA = 0;
  let normB = 0;

  for (let i = 0; i < a.length; i++) {
            sum += a[i] * b[i];
            normA += a[i] ** 2;
            normB += b[i] ** 2;
            console.log(a[1])
  }

  normA = Math.sqrt(normA);
  normB = Math.sqrt(normB);

  return normA === 0 || normB === 0 ? 0 : sum / (normA * normB);
}
```

Теоретичні відомості

Для реалізації програмного забезпечення відповідно до поставленого завдання, скористаємося мовою програмування JavaScript з простим графічним інтерфейсом. В якості міри подібності документу до пошукового запиту використаємо косинусну міру. За граничне значення подібності документів було обрано число: 0,15.

Результати виконання роботи

Vector Space Model

Documents

document 1: "This is the first document"
document 2: "BMW or Opel, Volvo or Nisan"
document 3: "This work is useful for many people"
document 4: "Each car brand has a right to life"
document 5: "Football players always show emotions after scoring a goal"

Search query

Search

Search query:

Each car brand has a right to life

Result:

similarity document 4: 1.000

Standard Boolean

Terms

term 1: "tulip"
term 2: "rose"
term 3: "chamomile"
term 4: "volvo"
term 5: "nisan"

Documents

document 1: "I have an a rose, a tulip, a volvo. I have an rose, an tulip, a chamomile."
document 2: "This is the first document BMW or Opel, Volvo or Nisan This work is useful for many people Each car brand has a right to life Football players always show emotions after scoring a goal"

Enter search query

Seach

search query:

rose OR volvo

Result:

Documents matching the search results: 1,2

Висновки

В ході виконання даної лабораторної роботи, ми ознайомилися з найбільш поширеними моделями подання документів та набули практичні навички з програмної реалізації теоретико-множинної (стандартної булевої) та алгебраїчної (векторно-просторової) моделей подання документів. Розробка програмного забезпечення виконувалась з використанням мови програмування JavaScript. Відповідно до визначеного варіанту було використано міру tf-idf для реалізації векторно просторової моделі. За граничне значення подібності документів, з використанням косинусної міри подібності документів до пошукового запиту, було обрано число: 0,15.