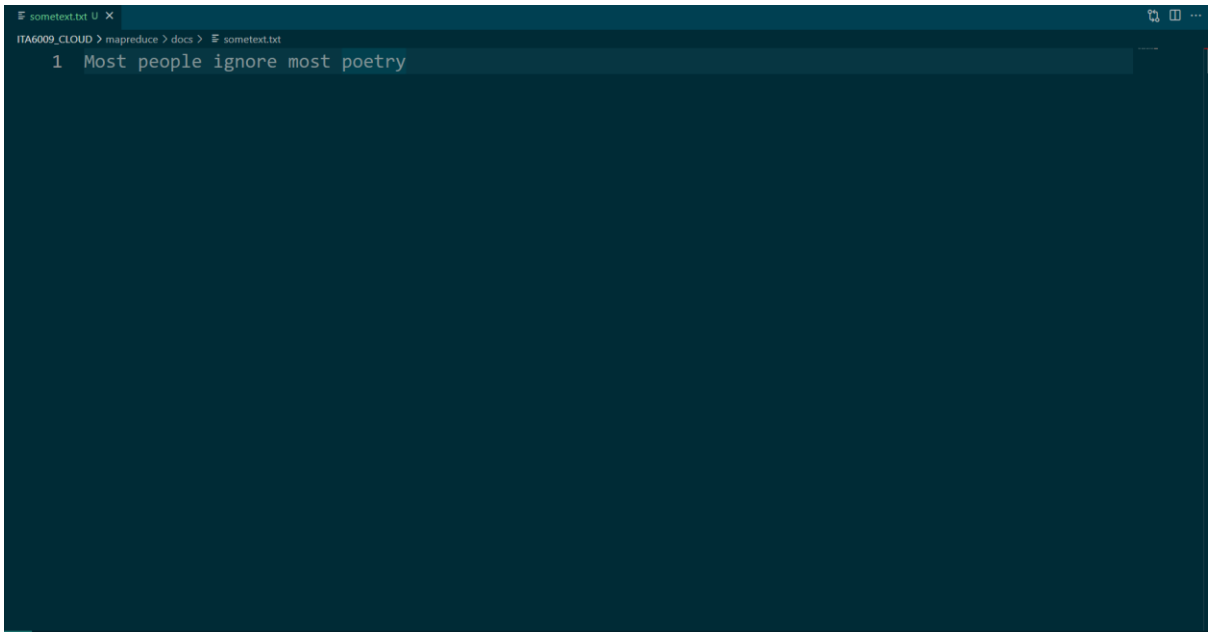
DIGITAL ASSIGNMENT 1

NAME: Kamran Ansari

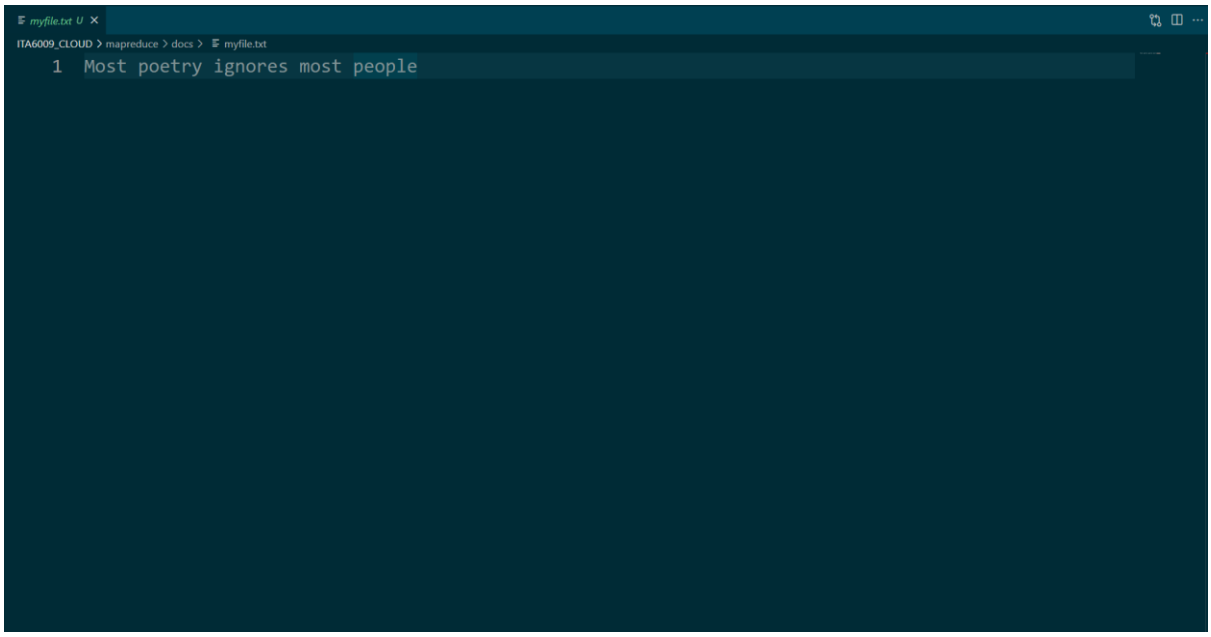
REG NO: 22MCA0223

There are two files in docs folder–

<u>sometext.txt</u>



<u>myfile.txt</u>



Problem 1: Counting the number of occurrences of each word in a collection of documents.

Code:

```
let fs = require("fs/promises");


const getFiles = async (source) =>
  (await fs.readdir(source, { withFileTypes: true }))
    .filter((dirent) => dirent.isFile())
    .map((dirent) => dirent.name);


const mapFunction = (content) => {
  const mapArray = [];
  const words = content.split(/[(\r\n)\s]/g).filter((s) => s.length);

  words.forEach((word) => {
    mapArray.push([word.toLowerCase(), 1]);
  });

  return mapArray;
};

const sortFunction = (map) => {
  return map.flat().sort((a, b) => (a[0][0] <= b[0][0] ? -1 : 1));
};

const groupFunction = (map) => {
  const groupedMap = new Map();

  map.forEach((word) => {
    if (!groupedMap.has(word[0])) {
      groupedMap.set(word[0], [1]);
    } else {
      groupedMap.set(word[0], groupedMap.get(word[0]).concat(1));
    }
```

```
  });

  return groupedMap;
};

const reduceFunction = (map) => {
  map.forEach((value, key) => {
    map.set(
      key,
      value.reduce((a, b) => a + b, 0)
    );
  });

  return map;
};

async function main() {
  const filesInDirectory = await getFiles("docs");

  const contentInFiles = await Promise.all(
    filesInDirectory.map((file) => {
      return fs.readFile(`docs/${file}`, {
        encoding: "utf-8",
      });
    })
  );

  const wordsMap = await Promise.all(
    contentInFiles.map(async (content) => {
      return mapFunction(content);
    })
  );

  console.log("\nAfter Map");
```

```
  console.log(wordsMap);

  const sortedMap = sortFunction(wordsMap);

  console.log("\nAfter Sorting");
  console.log(sortedMap);

  const groupedMap = groupFunction(sortedMap);

  console.log("\nAfter Grouping");
  console.log(groupedMap);

  const reducedMap = reduceFunction(groupedMap);

  console.log("\nAfter Reducing");
  console.log(reducedMap);
}

main();
```

Output:

```
deadmercury@DESKTOP-QUKDS0U MINGW64 /e/LEARNING/vit_mca/ITA6009_CLOUD/mapreduce (main)
$ node Q1.js

After Map
[
  [
    [ 'most', 1 ],
    [ 'poetry', 1 ],
    [ 'ignores', 1 ],
    [ 'most', 1 ],
    [ 'people', 1 ]
  ],
  [
    [ 'most', 1 ],
    [ 'people', 1 ],
    [ 'ignore', 1 ],
    [ 'most', 1 ],
    [ 'poetry', 1 ]
  ]
]

After Sorting
[
  [ 'ignore', 1 ],
  [ 'ignores', 1 ],
  [ 'most', 1 ],
```

```
  [ 'most', 1 ],
  [ 'poetry', 1 ],
  [ 'people', 1 ],
  [ 'people', 1 ],
  [ 'poetry', 1 ]
]

After Grouping
Map(5) {
  'ignore' => [ 1 ],
  'ignores' => [ 1 ],
  'most' => [ 1, 1, 1, 1 ],
  'poetry' => [ 1, 1 ],
  'people' => [ 1, 1 ]
}

After Reducing
Map(5) {
  'ignore' => 1,
  'ignores' => 1,
  'most' => 4,
  'poetry' => 2,
  'people' => 2
}
```

Problem 2: Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents.

Code:

```
let fs = require("fs/promises");


const getFiles = async (source) =>
  (await fs.readdir(source, { withFileTypes: true }))
    .filter((dirent) => dirent.isFile())
    .map((dirent) => dirent.name);


const mapFunction = (content) => {
  const mapArray = [];
  const words = content.split(/[(\r\n)\s]/g).filter((s) => s.length);

  words.forEach((word) => {
    mapArray.push([word.length, word]);
  });

  return mapArray;
};

const sortFunction = (map) => {
  return map.flat().sort((a, b) => a[0] - b[0]);
};

const groupFunction = (map) => {
  const groupedMap = new Map();

  map.forEach((word) => {
    if (!groupedMap.has(word[0])) {
      groupedMap.set(word[0], [word[1]]);
    } else {
      groupedMap.set(word[0], groupedMap.get(word[0]).concat(word[1]));
    }
```

```
  });

  return groupedMap;
};

const reduceFunction = (map) => {
  map.forEach((value, key) => {
    map.set(
      key,
      value.reduce((a, b) => a + 1, 0)
    );
  });

  return map;
};

async function main() {
  const filesInDirectory = await getFiles("docs");

  const contentInFiles = await Promise.all(
    filesInDirectory.map((file) => {
      return fs.readFile(`docs/${file}`, {
        encoding: "utf-8",
      });
    })
  );

  const wordsMap = await Promise.all(
    contentInFiles.map(async (content) => {
      return mapFunction(content);
    })
  );

  console.log("\nAfter Map");
```

```
  console.log(wordsMap);

  const sortedMap = sortFunction(wordsMap);

  console.log("\nAfter Sorting");
  console.log(sortedMap);

  const groupedMap = groupFunction(sortedMap);

  console.log("\nAfter Grouping");
  console.log(groupedMap);

  const reducedMap = reduceFunction(groupedMap);

  console.log("\nAfter Reducing");
  console.log(reducedMap);
}

main();
```

Output:

```
deadmercury@DESKTOP-QUKDS0U MINGW64 /e/LEARNING/vit_mca/ITA6009_CLOUD/mapreduce (main)
$ node Q2.js

After Map
[
  [
    [ 4, 'Most' ],
    [ 6, 'poetry' ],
    [ 7, 'ignores' ],
    [ 4, 'most' ],
    [ 6, 'people' ]
  ],
  [
    [ 4, 'Most' ],
    [ 6, 'people' ],
    [ 6, 'ignore' ],
    [ 4, 'most' ],
    [ 6, 'poetry' ]
  ]
]
```

```
After Sorting
[
  [ 4, 'Most' ],
  [ 4, 'most' ],
  [ 4, 'Most' ],
  [ 4, 'most' ],
  [ 6, 'poetry' ],
  [ 6, 'people' ],
  [ 6, 'people' ],
  [ 6, 'ignore' ],
  [ 6, 'poetry' ],
  [ 7, 'ignores' ]
]

After Grouping
Map(3) {
  4 => [ 'Most', 'most', 'Most', 'most' ],
  6 => [ 'poetry', 'people', 'people', 'ignore', 'poetry' ],
  7 => [ 'ignores' ]
}

After Reducing
Map(3) { 4 => 4, 6 => 5, 7 => 1 }
```

Problem 3: Counting the number of occurrences of anagrams in a collection of documents. Anagrams are words with the same set of letters but in a different order (e.g., the words “listen” and “silent”).

Code:

```
let fs = require("fs/promises");


const getFiles = async (source) =>
  (await fs.readdir(source, { withFileTypes: true }))
    .filter((dirent) => dirent.isFile())
    .map((dirent) => dirent.name);


const mapFunction = (content) => {
  const mapArray = [];
  const words = content.split(/[(\r\n)\s]/g).filter((s) => s.length);


  words.forEach((word) => {
    mapArray.push([word.toLowerCase().split("").sort().join(""), [1]]);
  });


  return mapArray;
};


const sortFunction = (map) => {
  return map.flat().sort((a, b) => (a[0][0] <= b[0][0] ? -1 : 1));
};


const groupFunction = (map) => {
  const groupedMap = new Map();


  map.forEach((word) => {
    if (!groupedMap.has(word[0])) {
      groupedMap.set(word[0], [1]);
    } else {
      groupedMap.set(word[0], groupedMap.get(word[0]).concat(1));
```

```
    }
  });

  return groupedMap;
};

const reduceFunction = (map) => {
  map.forEach((value, key) => {
    map.set(
      key,
      value.reduce((a, b) => a + b, 0)
    );
  });

  return map;
};

async function main() {
  const filesInDirectory = await getFiles("docs");

  const contentInFiles = await Promise.all(
    filesInDirectory.map((file) => {
      return fs.readFile(`docs/${file}`, {
        encoding: "utf-8",
      });
    })
  );

  const wordsMap = await Promise.all(
    contentInFiles.map(async (content) => {
      return mapFunction(content);
    })
  );
```

```
  console.log("\nAfter Map");
  console.log(wordsMap);

  const sortedMap = sortFunction(wordsMap);

  console.log("\nAfter Sorting");
  console.log(sortedMap);

  const groupedMap = groupFunction(sortedMap);

  console.log("\nAfter Grouping");
  console.log(groupedMap);

  const reducedMap = reduceFunction(groupedMap);

  console.log("\nAfter Reducing");
  console.log(reducedMap);
}

main();
```

Output:

```
deadmercury@DESKTOP-QUKDS0U MINGW64 /e/LEARNING/vit_mca/ITA6009_CLOUD/mapreduce (main)
$ node Q3.js

After Map
[
  [
    [ 'most', [Array] ],
    [ 'eoprty', [Array] ],
    [ 'eginors', [Array] ],
    [ 'most', [Array] ],
    [ 'eelopp', [Array] ]
  ],
  [
    [ 'most', [Array] ],
    [ 'eelopp', [Array] ],
    [ 'eginor', [Array] ],
    [ 'most', [Array] ],
    [ 'eoprty', [Array] ]
  ]
]
```

```
After Sorting
[
  [ 'eoprty', [ 1 ] ],
  [ 'eginor', [ 1 ] ],
  [ 'eelopp', [ 1 ] ],
  [ 'eelopp', [ 1 ] ],
  [ 'eginors', [ 1 ] ],
  [ 'eoprty', [ 1 ] ],
  [ 'most', [ 1 ] ],
  [ 'most', [ 1 ] ],
  [ 'most', [ 1 ] ],
  [ 'most', [ 1 ] ]
]

After Grouping
Map(5) {
  'eoprty' => [ 1, 1 ],
  'eginor' => [ 1 ],
  'eelopp' => [ 1, 1 ],
  'eginors' => [ 1 ],
  'most' => [ 1, 1, 1, 1 ]
}
```

```
After Reducing
Map(5) {
  'eoprty' => 2,
  'eginor' => 1,
  'eelopp' => 2,
  'eginors' => 1,
  'most' => 4
}
```