Прізвище:
Дацишин
Ім'я: Роман
Група: КН-405
Варіант: 4

Кафедра.: Кафедра Систем
Автоматизованого Проектування
Дисципліна: Дискретні моделі в САПР
Перевірив: Кривий Р.З.

Звіт
До лабораторної роботи №3
На тему "Алгоритм рішення задачі комівояжера"

Мета роботи: Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі комівояжера.

Короткі теоретичні відомості:

Можна знайти точний розв'язок задачі комівояжера, тобто, обчислити довжини всіх можливих маршрутів та обрати маршрут з найменшою довжиною.

Однак, навіть для невеликої кількості міст в такий спосіб задача практично нерозв'язна. Для простого варіанта, симетричної задачі з n містами, існує (n − 1)!/ 2 можливих маршрутів, тобто, для 15 міст існує 43 мільярдів маршрутів та для 18 міст вже 177 більйонів. Те, як стрімко зростає тривалість обчислень можна показати в наступному прикладі. Якщо існував би пристрій, що знаходив би розв'язок для 30 міст за годину, то для для двох додаткових міст в тисячу раз більше часу; тобто, більш ніж 40 діб.

Відомо багато різних методів рішення задачі комівояжера. Серед них можна виділити методи розроблені Белмором і Немхаузером, Гарфинкелем і Немхаузером, Хелдом і Карном, Стекханом. Всі ці методи відносяться до одного з двох класів: а) методи рішення, які завжди приводять до знаходження оптимального рішення, але потребують для цього, в найгіршому випадку, недопустимо великої кількості операцій(метод гілок та границь); б) методи, які не завжди приводять до находження оптимального результату, але потребують для цього допустимої великої кількості операцій (метод послідовного покращення рішення).

Індивідуальне завдання:

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на виконання програму відповідного методу.
4. Проглянути результат роботи програм. Результат роботи може бути позитивним (шлях знайдено) або негативним (шлях відсутній).
5. У випадку, коли шлях знайдено (не знайдено), необхідно модифікувати граф, коректуючи два або три зв'язки, щоб знайти такий граф, на якому задача комівояжера не вирішується (вирішується).
6. Порівняти результати, отримані за допомогою різних алгоритмів і зробити висновок.
7. Зафіксувати результати роботи у викладача.
8. Оформити і захистити звіт.

Вхідні дані (файл 1):

```
6
0 0 69 60 10 20
0 0 0 31 39 2
69 0 0 0 59 0
60 31 0 0 0 36
10 39 59 0 0 79
20 2 0 36 79 0
```

Було написано програму для вирішення задачі комівояжера методом гілок та границь. Код програми , а також результат її виконання наведено нижче

Код програми

```
let initialString = "";
let matrixLength;
let html = "";
let m1 = [];

let input = document.querySelector("input");

input.addEventListener("change", () => {
  let files = input.files;

  if (files.length == 0) return;

  const file = files[0];

  let reader = new FileReader();

  reader.onload = (e) => {
    let file = e.target.result;
    file = file.replace(/(\r\n|\n|\r)/gm, " ");

    initialString = file;
    parseMatrix(initialString);
  };

  reader.onerror = (e) => alert(e.target.error.name);

  reader.readAsText(file);
});

const parseMatrix = (matrix) => {
  matrixLength = matrix[0];
  let str = matrix.substring(2);

  let arr = [];
  let num = "";

  for (let i = 0; i < str.length; i++) {
    if (str[i] !== " ") {
      num += str[i];
    } else {
      arr.push(parseInt(num));
      num = "";
    }
  }

  let m = [];
  for (let i = 0; i < matrixLength; i++) {
    m[i] = new Array(matrixLength);
  }

  let j = 0;
  let k = 0;

  for (let i = 0; i < arr.length; i++) {
    if (j < Math.sqrt(arr.length)) {
      m[k][j] = arr[i];
```

```
      j++;
    } else {
      j = 0;
      k++;
      m[k][j] = arr[i];
      j++;
    }
  }

  html += `<br/><h1>Open a terminal</h1><br/>`;
  document.getElementById("container").innerHTML = html;
  commisVoyageur(graphFiller(m));
};

function commisVoyageur(arr) {
  console.log(arr);

  let arrCopy = arr.map(function (arr1) {
    return arr1.slice();
  });

  arr = minColRowDel(arr)[0];
  let minRow = minColRowDel(arr)[1];
  let minCol = minColRowDel(arr)[2];
  console.log(arr);

  let minLim =
    minRow.reduce((a, b) => a + b, 0) + minCol.reduce((a, b) => a + b, 0);
  console.log(`minLim: ${minLim}`);

  let limit = 0;
  let banList = [];
  let result = `Path: `;

  while (limit < 20 && banList.length < arr.length * 2) {
    limit++;

    let maxZeroMatrix = maxZeroMatrixCount(arr);
    console.log(maxZeroMatrix);
    let maxZero = {
      value: 0,
      position: [0, 0],
    };
    for (let i = 0; i < maxZeroMatrix.length; i++) {
      for (let j = 0; j < maxZeroMatrix[0].length; j++) {
        if (maxZeroMatrix[i][j] > maxZero.value) {
          maxZero.value = maxZeroMatrix[i][j];
          maxZero.position = [i, j];
        }
      }
    }
    console.log(maxZero);

    includeResult = include(arr, maxZero.position, banList);
    console.log(includeResult);
    notIncludeResult = notInclude(arr, maxZero.position);
    console.log(notIncludeResult);

    if (includeResult.minLim < notIncludeResult.minLim) {
      console.log(`\n\nIncluding (${maxZero.position})\n\n`);

      arr = includeResult.matrix;
```

```
    banList.push(maxZero.position[0], maxZero.position[1]);
  } else {
    console.log(`\n\nNot including (${maxZero.position})\n\n`);
    arr = notIncludeResult.matrix;
  }

  console.log("BanList:" + banList);

  tempResult = "Edges list: ";
  for (let i = 0; i < banList.length - 1; i += 2) {
    tempResult += `(${banList[i]},${banList[i + 1]})`;
    if (i != banList.length - 2) tempResult += ` => `;
  }
}

for (let i = 0; i < banList.length - 1; i += 2) {
  const element = banList[i];
  result += `(${banList[i]}, ${banList[i + 1]}) => `;
}

console.log(result);

let sum = 0;

for (let i = 0; i < arrCopy.length; i++) {
  for (let j = 0; j < arrCopy[i].length; j++) {
    for (let k = 0; k < banList.length - 1; k += 2) {
      if (i == banList[k] && j == banList[k + 1]) sum += arrCopy[i][j];
    }
  }
}

console.log("SUM: " + sum);

let resultCycle = cycleBuilder(banList);
console.log(resultCycle);

function minColRowDel(arr) {
  let tempArr = arr.map(function (arr) {
    return arr.slice();
  });

  let minRow = [];
  let minCol = [];

  for (let i = 0; i < tempArr.length; i++) {
    minRow.push(Infinity);
    for (let j = 0; j < tempArr[i].length; j++) {
      if (tempArr[i][j] < minRow[i]) minRow[i] = tempArr[i][j];
    }
  }

  console.log(`minRow: ${minRow}`);

  for (let i = 0; i < tempArr.length; i++) {
    for (let j = 0; j < tempArr[i].length; j++) {
      tempArr[i][j] -= minRow[i];
    }
  }

  console.log(tempArr);
```

```
  for (let i = 0; i < tempArr.length; i++) {
    minCol.push(Infinity);
    for (let j = 0; j < tempArr[i].length; j++) {
      if (tempArr[j][i] < minCol[i]) minCol[i] = tempArr[j][i];
    }
  }

  console.log(`minCol: ${minCol}`);

  for (let i = 0; i < tempArr.length; i++) {
    for (let j = 0; j < tempArr[i].length; j++) {
      tempArr[j][i] -= minCol[i];
    }
  }

  return [tempArr,
minRow, minCol];
```

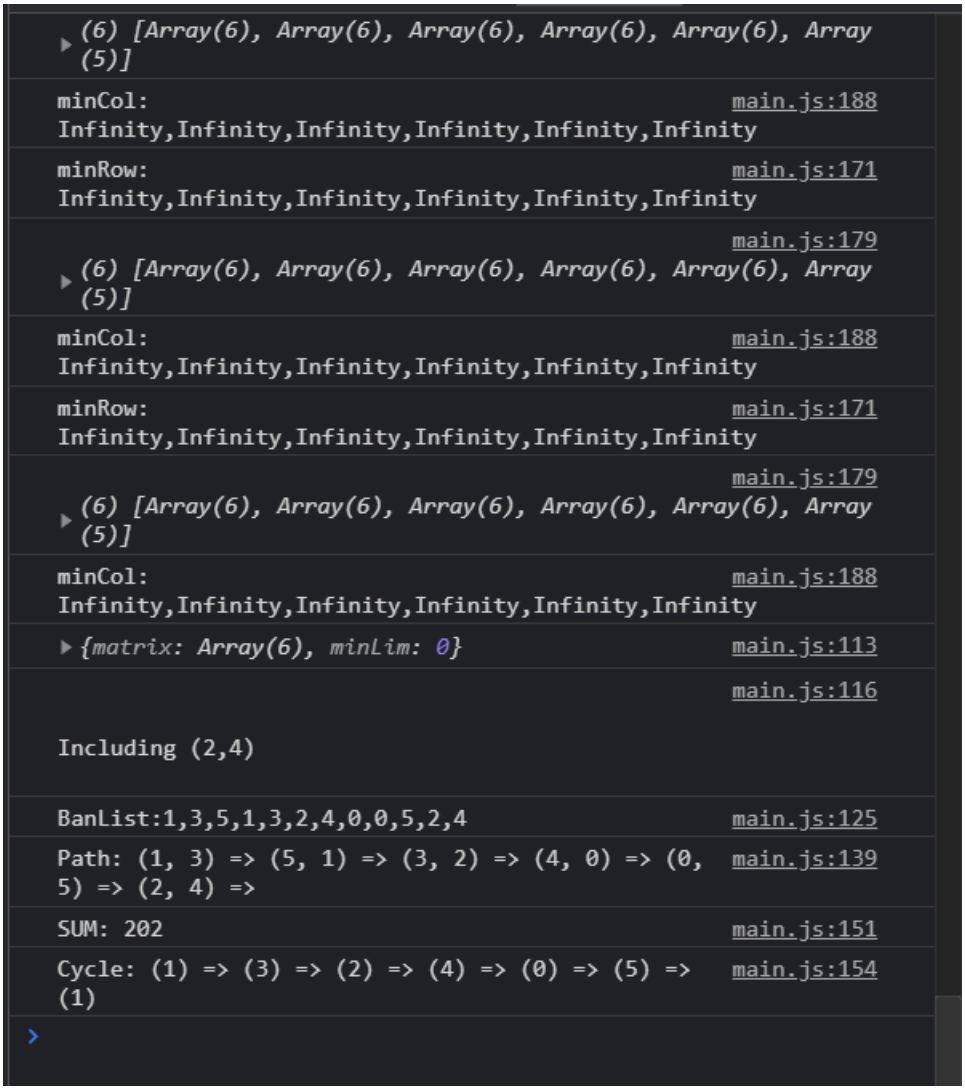

Рис. 1. Результат виконання програми

Пояснення до програми

Оскільки програмно реалізувати метод гілок та меж для неповного графа виявилося дуже важко, тому моя програма заміняє відсутні ребра (0) на ребра з вагою (найбільша вага ребра в графі + 1), ці ребра можна буде використовувати, але пріоритет в алгоритмі у них буде нижчий.

Повна версія коду доступна на GitHub:
https://github.com/RomanDatsyshyn/DiscreteModelsLabs

Висновок: під час виконання цієї лабораторної роботи я ознайомився з алгоритмами рішення задачі комівояжера та реалізував програмно один з них, а саме написав програму для вирішення даної задачі методом гілок та меж.