Ministerul Educaţiei, Culturii și Cercetării  
Universitatea Tehnică a Moldovei

Departamentul Informatică și Ingineria Sistemelor

**RAPORT**

Lucrarea de laborator nr.4  
la Analiza și Proiectarea Algoritmilor

A efectuat:   
st. gr. TI-206 Borș Nicoleta

A verificat: Ernest Bîtca

Chişinău – 2021

**Tema**: Metoda programării dinamice

**Scopul lucrării:**

1. Studierea metodei programării dinamice.
2. Analiza şi implementarea algoritmilor de programare dinamică.
3. Compararea tehnicii greedy cu metoda de programare dinamică.

**Sarcina:**

1. De studiat metoda programării dinamice de proiectare a algoritmilor.
2. De implementat într-un limbaj de programare algoritmii prezentaţi mai sus.
3. De făcut analiza empirică a acestor algoritmi pentru un graf rar şi pentru un graf dens.
4. De alcătuit un raport.

#### **Întrebări de control:**

#### **Descrieţi metoda programare dinamică.**

#### Programarea dinamică este în esenţă un proces decizional în mai multe etape: în starea iniţială a problemei luăm prima decizie, care determină o nouă stare a problemei în care luăm o decizie.

#### **De ce această metodă se numeşte programare dinamică?**

#### Deoarece problema este rezolvată în etape dependente de timp, astfel este dinamică.

#### **Care este diferenţa între metoda divide et impera şi metoda programării dinamice?**

#### Programarea dinamică presupune rezolvarea unei probleme prin descompunea ei în subprobleme și rezolvarea acestora. Spre deosebire de divide et impera, subprogramele nu sunt disjuncte, ci se suprapun. Metoda divide et impera se poate aplica problemelor care permit descompunerea lor în subprograme independente și obținem 3 faze principale: divide, stăpânește și combină.

#### **Care este clasificarea problemelor de programare dinamică?**

#### Există mai multe clasificări ale problemelor de programare dinamică:

#### 1. După natura deciziilor pot fi deterministe sau nedeterministe.

#### 2. După valoarea deciziilor pot fi de optimizare, de neoptimizare.

#### 3. După numărul de dimensiuni pot fi unidimensionale, sau multidimensionale.

#### 4. După numărul deciziilor care se pot lua la un pas pot fi unidecizionale, sau multidecizionale.

#### 5. După direcţia deciziilor putem avea programare dinamică înainte sau programare dinamică înapoi.

#### **Ce se întâmplă în cazul când costurile arcelor grafurilor prelucrate cu algoritmii Dijkstra şi Floyd sunt negative?**

#### Floyd lucrează și la grafurile cu costuri negative, pe când Dijkstra nu.

#### **Tabelul 1** – Timpul (ms) de execuție pentru Algoritmul Dijkstra și Floyd pentru un graf **dens**, dintr-un nod sursa în celelalte noduri.

|  |  |  |  |
| --- | --- | --- | --- |
| **Graf dens** | Dijkstra (un nod sursă) | Dijkstra | Floyd |
| n = 10 | 0.15 | 0.5 | 0.09 |
| n = 50 | 0.69 | 6.39 | 4.30 |
| n = 100 | 4.30 | 17.80 | 7.40 |
| n = 250 | 4.60 | 133.30 | 59.5 |
| n = 500 | 6.69 | 950 | 333.89 |
| n = 1000 | 12.80 | 7184 | 2815.69 |

#### **Tabelul 2** – Timpul (ms) de execuție pentru Algoritmul Dijkstra și Floyd pentru un graf **rar**, dintr-un nod sursa în celelalte noduri.

|  |  |  |  |
| --- | --- | --- | --- |
| **Graf rar** | Dijkstra (un nod sursă) | Dijkstra | Floyd |
| n = 10 | 0.15 | 0.2 | 0.15 |
| n = 50 | 0.39 | 7.10 | 4.39 |
| n = 100 | 0.79 | 13.29 | 6.79 |
| n = 250 | 4.80 | 121.30 | 43.80 |
| n = 500 | 6.30 | 836.10 | 367.80 |
| n = 1000 | 15.0 | 7278.5 | 2542.30 |

#### Din graficul de mai sus, observăm ca algoritmul Dijkstra este cu mult mai lent decât Floyd atunci când avem nevoie de a afla datele tuturor nodurilor. Dacă avem nevoie să aflăm datele doar dintr-un nod sursa, atunci Dijkstra este cu mult mai rapid decât Floyd.

#### Observăm că algoritmul Dijkstra nu este mai de loc influențat e tipul grafului. Algoritmul Floyd însă este un pic mai lent atunci când numărul de vârfuri este mai mare și graful este dens însă diferențele nu sunt foarte mari.

#### 

|  |  |  |
| --- | --- | --- |
| Complexitatea | Pentru un nod | Pentru toate nodurile |
| Dijkstra | O() | O() |
| Floyd | O() | O() |

#### **Concluzie:**

#### În cadrul lucrării de laborator la tema “Metoda programării dinamice” am avut posibilitatea să compar doi algoritmi Dijkra și Floyd, unul facând parte din algoritmii Greedy iar celălalt din Metoda programării dinamice. Am observat că Dijkstra ne arată drumul minim de la un nod la toate nodurile pe când Floyd ne arată drumurile de la oricare nod spre celelalte.

#### Comparând timpul de execuție în funcție de tipul grafului, dens sau rar, am observat că ambii algoritmi nu sunt foarte tare influențați. Pentru a rezolva o problemă în care avem nevoie să căutam drumurile de la toate vârfurile, la celelalte, algoritmul lui Floyd, datorită simplităţii lui este mai rapid decât Dijkstra. În cazul în care avem nevoie de datele dintr-un singur nod sursa în celelalte noduri algoritmul Dijkstra este cu mult mai rapid decât Floyd.

#### **Bibliografia:**

#### Else APA16.3 - <https://else.fcim.utm.md/course/view.php?id=773>

#### Else APA16.1 - <https://else.fcim.utm.md/course/view.php?id=7>

#### GitHub - <https://github.com/Nicoleta-Bors/APA.git>

#### Floyd–Warshall algorithm - <https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm>

#### Dijkstra's algorithm - <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>

#### **Anexa 1 – Codul programului**

let graf = [], n;

const unGraf = () => {

  graf = [

    [0, 2, 3, Infinity],

    [3, 0, 4, Infinity],

    [Infinity, 2, 0, 4],

    [Infinity, Infinity, 4, 0],

  ];

  n = graf.length;

};

const initiereMatrice = (n, complete) => {

  for (let i = 0; i < n; i++) {

    graf[i] = [];

    for (let j = 0; j < n; j++) {

      graf[i][j] = complete;

    }

  }

};

const grafRar = () => {

  let random1, random2;

  initiereMatrice(n, Infinity);

  for (let i = 0; i < n; i++) graf[i][i] = 0;

  for (let i = 0; i < n - 1; i++) {

    graf[i][i + 1] = Math.floor(Math.random() \* n) + 1;

    graf[i + 1][i] = Math.floor(Math.random() \* n) + 1;

    random1 = Math.floor(Math.random() \* (n - 1));

    random2 = Math.floor(Math.random() \* (n - 1));

    if (random1 !== random2) {

      graf[random1][random2] = Math.floor(Math.random() \* n) + 1;

    }

  }

};

const grafDens = () => {

  initiereMatrice(n, Infinity);

  for (let i = 0; i < n; i++) {

    for (let j = 0; j < n; j++) {

      if (i === j) {

        graf[i][j] = 0;

      } else {

        graf[i][j] = Math.floor(Math.random() \* n) + 1;

      }

    }

  }

};

*// ---------------------------------------------------------------- Dijkstra*

const minDistance = (distante, varfuriVizitate) => {

  let min = Infinity;

  let minIndex = -1;

  for (let i = 0; i < n; i++) {

    if (varfuriVizitate[i] == false && distante[i] <= min) {

      min = distante[i];

      minIndex = i;

    }

  }

  return minIndex;

};

const dijkstra = (graf, src) => {

  console.log('Distantele din nodul ' + src);

  let distante = new Array(n);

  let varfuriVizitate = new Array(n);

  for (let i = 0; i < n; i++) {

    distante[i] = Infinity;

    varfuriVizitate[i] = false;

  }

  distante[src] = 0;

  let count = 0;

  while (count < n - 1) {

    let i = minDistance(distante, varfuriVizitate);

    varfuriVizitate[i] = true;

    for (let j = 0; j < n; j++) {

      if (!varfuriVizitate[j] &&

        graf[i][j] != Infinity &&

        graf[i][j] != 0 &&

        distante[i] != Infinity) {

        if (distante[i] + graf[i][j] < distante[j]) {

          distante[j] = distante[i] + graf[i][j];

        }

      }

    }

    count++;

  }

  for (let i = 0; i < n; i++) console.log(`[${src}][${i}] = ${distante[i]}`);

};

*// ---------------------------------------------------------------- Floyd*

const floyd = (graf) => {

  let distante = [...graf];

  let i, j, k;

  for (k = 0; k < n; k++) {

    for (i = 0; i < n; i++) {

      for (j = 0; j < n; j++) {

        if (distante[i][j] > distante[i][k] + distante[k][j]) {

          distante[i][j] = distante[i][k] + distante[k][j];

        }

      }

    }

  }

  for (let i = 0; i < n; i++) {

    console.log('Distantele din nodul ' + i);

    for (let j = 0; j < n; j++) console.log(`[${i}][${j}] = ${distante[i][j]}`);

  }

};

unGraf();

*// n = 10;*

*// grafDens();*

*// grafRar();*

const startTime = window.performance.now();

*// for (let i = 0; i < n; i++) dijkstra(graf, i);*

*// dijkstra(graf, 0);*

floyd(graf);

const endTime = window.performance.now();

console.log(`%cTimpul de rulare: ${endTime - startTime} ms = ${(endTime - startTime) / 1000} s`,'color: #E05297');