Informatica: C# WPO 10

1. Inhoud

2D arrays, lijsten van arrays, NULL-values

2. Oefeningen

- Demo 1: Fill and print 2D array
- Demo 2: Fill and print list of array
- A: Matrix optelling
- A: Matrix * constante
- A: Picasso
- A: Vergelijk matrices
- A: Matrix * vector
- A: Matrix * matrix
- E: Stringsplit
- E: Bovendriehoeksmatrix
- E: Eliminatie van Gauss
- X: Sudoku

2.1 Demo 1: Fill and print 2D array

Schrijf een programma waarin je een 2D array aanmaakt van integers. Gebruik 2 for loops om de array in te vullen met random waarden.

2.2 Demo 2: Fill and print list of array

Schrijf een programma waarin je een list van arrays bijhoudt. Vul de list aan met arrays met lengte 1 t.e.m. 10. Print vervolgens de list of array af in de console. De console is een alternatief voor de textbox en kan enkel gebruikt worden als debug output.

2.3 A: Matrix optelling

Schrijf een programma waarin je 2 matrices met elkaar kan optellen. Matrices kunnen gezien worden als 2D arrays. Je mag hierbij volgende matrices beschouwen:

$$M_1 = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 10 & 12 \\ 0 & 8 & 2 \end{bmatrix} \tag{1}$$

$$M_2 = \begin{bmatrix} -1 & -10 & 3 \\ 0 & 102 & -12 \\ 2 & -68 & -25 \end{bmatrix} \tag{2}$$

$$M_2 = \begin{bmatrix} -10 & -1 & 68 \\ -5 & 7 & -2 \\ 3 & -8 & -5 \end{bmatrix} \tag{3}$$

Je mag hierbij zelf kiezen of je de som van de matricies in de console of in een textbox afprint. Schrijf een aparte functie die de optelling tussen 2 matrices uitvoert. De functie retourneert de resultaatsmatrix en heeft als argumenten de 2 op te tellen matrices.
Opgelet: beide matrices kunnen enkel opgeteld worden indien de dimensies exact overeen komen.

2.4 A: Matrix * constante

Voor deze opgave mag je de matrices van voorgaande opgaven beschouwen. Deze keer ga je de matrices vermenigvuldigen met een bepaalde constante. De constanten die je kan gebruiken zijn:

- 3,
- 10,
- -5,
- 17,
- 3.

Schrijf een functie die de vermenigvuldiging uitvoert. De functie retourneert de resultaatmatrix en heeft als argumenten de originele matrix en de te vermenigvuldigen constante.

2.5 A: Picasso

Schrijf een programma dat een 2D array met random getallen invult en de array a.d.h.v. kleine rechthoeken op een canvas weergeeft. Schrijf een aparte functie die een 2D-array van 100 bij 100 elementen met randomwaarden tussen 0 en 255 invult. Een andere functie neemt deze matrix op als argument en tekent (in grijswaarden) alle waarden a.d.h.v. kleine vierkanten op een canvas. Merk op dat de volledige canvas gevuld dient te worden. Dus zorg voor de juiste schaling wat betreft de hoogte en breedte van de kleine rechthoeken.

2.6 A: Vergelijk matrices

Schrijf een programma waarin je 2 matrices met elkaar gaat vergelijken. Je mag hierbij de matrices van hierboven gebruiken. Schrijf een functie die de vergelijking uitvoert. Merk op dat hierbij alle cellen (elementen) van beide matrices aan elkaar vergeleken moeten worden. Het is enkel indien alle elementen overeenkomen dat de matrices gelijk zijn.
Opgelet: beide matrices kunnen enkel opgeteld worden indien de dimensies exact overeen komen.

2.7 A: Matrix * vector

Herneem voorgaande opgave en schrijf nu een functie die een matrix met een vector kan vermenigvuldigen. Schrijf hierbij eerst op papier uit hoe de vermenigvulding plaatsvindt. Daarna schrijf je hiervoor een functie die een 1D-array retourneert en 2D array en 1D array als argumenten opneemt. Aan welke voorwaarden moeten de dimensies van de array/vector voldoen opdat de vermenigvuldiging plaatsvindt. Print het resultaat af in de console. Je mag hierbij zelf de vectoren (1D arrays) kiezen. Je kan hierbij ook de matrices van een voorgaande opgave hernemen.

2.8 A: Matrix * matrix

Deze opgave is vrij gelijkaardig aan de vermenigvuldiging van een matrix met een vector. Hierbij zal je echter een matrix met een andere matrix vermenigvuldigen. Schrijf eerst op papier uit hoe de vermenigvuldiging plaatsvindt alvorens die te programmeren in een functie. Aan welke voorwaarden moeten de dimensies van de matrices (arrays) voldoen opdat de vermenigvuldiging plaatsvindt. Print het resultaat af in de console. Je kan hierbij ook de matrices van een voorgaande opgave hernemen.

2.9 E: Stringsplit

Schrijf zelf een functie die een string kan opdelen a.d.h.v. een karakter. Als argumenten krijgt deze functie een string en een karakter mee. De functie retourneert een lijst van strings. In deze opgave ga je volgt te werk.

1. Maak een variabele tempstring aan die je leeg laat.

2. Vergelijk het huidige karakter met de te vergelijken karakter van in de opgave. Zijn beide karakters gelijk, dan voeg je de huidige tempstring toe aan de lijst. Op dat moment maak je een nieuwe tempstring aan die je leeg maakt. Indien niet voeg je het huidige karakter toe aan de tempstring.

3. Op het einde zal je een nog een volle tempstring hebben. De string eindigt daarom niet altijd op het te zoeken karakter. Voeg daarom na de laatste iteratie de tempstring toe aan de lijst.

4. Retourneer de lijst. Merk op dat dit alleen mag indien tempstring niet leeg is.

2.10 E: Bovendriehoeksmatrix

In deze opgave ga je onderstaande matrix herleiden naar een bovendriehoeksmatrix.

$$V = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 2 \\ 1 & 2 & -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 2 \end{pmatrix} \tag{4}$$

De oplossing van deze vergelijking zou er als volgt moeten uitzien (na herleiden naar bovendriehoeksmatrix).

$$V = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 0 & 0 & -5 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 10 \end{pmatrix} \tag{5}$$

Hiervoor ga je als volgt te werk.

1. Hou rij 1 vast. Sla het eerste element van deze rij in een tijdelijke variabele op.
2. Ga op rij 2 staan en schaal de tijdedelijke variabele met element 1 van rij 2.
3. Vermenigvuldig de eerste rij met de geschaalde waarde en trek deze af van rij 2. Doe dit ook voor de kolomvector na het gelijkheidsteken.

De vergelijking zou na de eerste stap er als volgt moeten uitzien:

$$V = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 1 & 2 & -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 2 \end{pmatrix} \tag{6}$$

Voer dezelfde operatie uit voor rij 3 (t.o.v. rij 1) De schaling zal hier uiteraard anders zijn. Eenmaal de 3de rij behandeld is, ziet de vergelijking er als volgt uit:

$$V = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 0 & 1 & -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 2 \end{pmatrix} \tag{7}$$

Bij grotere matrices wordt deze operatie doorlopen totdat alle rijen behandeld zijn. De volledige operatie wordt herhaald voor de 2de kolom. De operatie start hier vanuit de 2de rij i.p.v. de eerste rij. Na de 2de kolom te hebben verwerkt zou de matrix er als volgt moeten uitzien:

$$V = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 0 & 0 & -5 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 10 \end{pmatrix} \tag{8}$$

Bij grotere matrices wordt de operatie herhaald totdat alle kolomen behandeld zijn. Er wordt dan telkens vanuit de diagonaalelement gestart en dan naar beneden gewerkt om de rijen uit te nullen.

2.11 E: Eliminatie van Gauss

De eliminatie van Gauss werkt op dezelfde manier als hierboven. Het enige verschil is dat de bovendriehoeksmatrix verder wordt opgelost naar een matrix waarin enkel waarden verschillend van 0 op de diagonaal staan. De niet diagonaal waarden zijn allemaal 0. Om dit te bereiken kan je als volgt te werk gaan.

- Neem de oplossing bekomen van vorige opgave (driehoeksmatrix).
- De methode werkt volledig omgekeerd aan het verkrijgen van een bovendriehoeksmatrix. Hou dus het uiterste rechtsonder element vast en elimineer de elementen van dezelfde kolom in de bovengelegen rijen.
- Herhaal deze methode voor de andere kolomen, telkens vertrekkende van de diagonaal (van rechts naar links).

Na deze operaties zou de oplossing er als volgt uitzien:

$$V = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -5 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 10 \end{pmatrix} \tag{9}$$

Het is mogelijk dat de waarden niet overeenkomen. De verhoudingen tussen de matrix en de kolomvector moeten voor elke rij dezelfde zijn als in de oplossing.

2.12 X: Sudoku - Opgelet - gebruik van recursie en backtracking - voor de echte durvers!

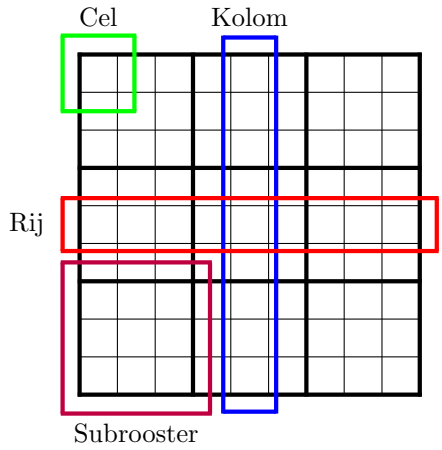
Backtracking omvat programmeerproblemen waarbij plausibele oplossingen onderzocht worden en niet-plausibele oplossingen genegeerd worden. Dit algoritme kan o.a. toegepast worden op puzzelproblemen (Sudoku, Achtkoninginnenprobleem,...), Minimum Spanning Trees (routering binnen netwerken), etc. Een bekende puzzel die hier opgelost zal worden met dit algoritme is de Sudoku. De klassieke Sudoku bestaat uit een rooster (Figuur 1) van 9 bij 9 cellen (81 elementen), waarbij volgende spelregels gehanteerd worden:

- in elke rij is elk getal uniek
- in elke kolom is elk getal uniek,
- elk subrooster van 3 bij 3 cellen bevat 9 unieke getallen.

De werkwijze van backtracking wordt hieronder beschreven.

- Ga naar de eerste lege cel in de Sudoku.
- Probeer de laagste waarde, hier 1, in te vullen. Ga naar de volgende lege cel indien aan alle spelregels voldaan zijn. Indien niet, incrementeer de celwaarde en controlleer opnieuw. Blijf dit herhalen totdat een mogelijke kandidaat gevonden is.

- Indien bij een verdere cel de maximale waarde overschreden wordt (hier 9), moet het algoritme terugkeren naar de vorige ingevulde cellen. Die cellen worden opgehoogd en het algoritme rekent opnieuw verder. Indien er geen oplossingen mogelijk zijn, zal het algoritme uit zichtzelf opnieuw naar de oorsprong begeven, waar het stopt met een foutmelding.



Figuur 1: Schematische voorstelling van een 9 bij 9 sudoku.

Programmeer een functie die toelaat om dergelijke puzzels op te lossen. Je mag hierbij sudokus uit een krant nemen en hardcoded programmeren in het programma.