Programmeermethoden

Arrays (vervolg)

Walter Kosters en Jonathan Vis

week 8: 30 oktober–3 november 2023

www.liacs.leidenuniv.nl/~kosterswa/pm/

week	onderwerp	boek	dictaat
30 okt–3 nov	Arrays (vervolg)	5	4.2,op31/36
6–10 nov	Arrays (vervolg 2)	5	4.2,op37/43
13–17 nov	Pointers	10	3.12,op44/46,52/55
20–24 nov	Recursie	13	3.10,op57/61
27 nov–1 dec	Datastructuren	17	5,op47/51
4–8 dec	Algoritmen, Python		
12–16 dec	Oude tentamens. . .		

op = opgaven uit opgavendictaat; zelf maken, antwoorden: zie website.

In rood: de weken met een deadline op de maandag erna.

Tentamen: woensdag 17 januari 2024, 13:00–16:00 uur; Universitair Sportcentrum. Denk aan het inschrijven!

Programmeermethoden

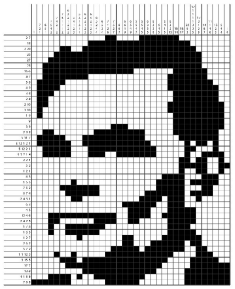
Home Algemeen Materiaal Opdrachten en Cijfers

Opdrachten en Cijfers

Programmeermethoden 2023
Derde programmeeropgave: Nonogram

De *derde* programmeeropgave van het vak **Programmeermethoden** in het najaar van 2023 heet *Nonogram*; zie ook het zevende werkcollege, achtste werkcollege en negende werkcollege, en lees geregeld deze pagina op WWW.

De opgave



Het is de bedoeling om een C++-programma te maken dat de gebruiker in staat stelt een Nonogram te maken en op te lossen via een menu-systeem. Dat betekent dat de gebruiker van het programma kan kiezen uit een aantal mogelijkheden, de zogeheten *opties*. Er is één submenu, waarin ook weer opties zijn. De bedoeling is dat de menukeuzes op één of twee regels staan, onder de puzzel (zie verderop).
De opties worden gekozen door de eerste letter van de betreffende optie in te toetsen (gevolgd door Enter), bijvoorbeeld een `s` of `S` om te stoppen. Uiteraard wordt een en ander duidelijk en ondubbelzinnig aan de gebruiker meegedeeld. Gebruik *geen* recursie!
Alle door de gebruiker ingetoetste symbolen moeten gecontroleerd worden, dat wil zeggen dat er binnen redelijke grenzen geen foute invoer geaccepteerd wordt. Zo zal het intoetsen van bijvoorbeeld `q` of `&` in het hoofdmenu genegeerd worden (tenzij er een optie `Q` is ...). Verder moet bij getalleninvoer karakter voor karakter ingelezen worden (met `cin.get ( );` als je elders ook nog `cin >> ...` gebruikt krijg je overigens soms problemen met "hangende Enter's"; gebruik dus overal `cin.get ( )`). Er moet ook op gelet worden dat er geen te grote getallen worden ingevoerd. Schrijf dus een geschikte functie `leesgetal` die de gelezen karakters (cijfers) omzet in een getal (tip: negeer alle "voorloop-Enter's"; verwerk alles tot en met de eerstvolgende enter, en maak hiervan zo goed mogelijk een getal, van een maximale grootte; zo kan `abc123defg999h`, als je een getal kleiner dan 10000 wilt, bijvoorbeeld verwerkt worden tot `1239`), en een functie `leesoptie` die netjes één karakter inleest en Enter's afhandelt! Aan de gebruiker mogen "redelijke" beperkingen worden gevraagd, bijvoorbeeld dat de in te voeren getallen maximaal vier cijfers hebben. Het programma moet dan echter wel bestand zijn tegen pogingen meer dan vier cijfers in te voeren. Ook het invoeren van letters in plaats van cijfers moet geen problemen opleveren. Houd het simpel!

Een *Nonogram*, ook bekend als een Japanse puzzel, en niet te verwarren met Sudoku, is een puzzel waarbij een zwart-wit plaatje moet worden gereconstrueerd uit zogeheten beschrijvingen. Het gaat om een rechthoekig *m* (rijen) bij *n* (kolommen) rooster. Elk vakje = pixel moet zwart (1/true) of wit (0/false) worden. Naast alle rijen en boven alle kolommen staat een rijtje gehele getallen: de lijn-beschrijving. Zo'n beschrijving geeft, in volgorde, de lengtes van de aaneengesloten rijtjes zwarte pixels aan. Zo betekent `3 1` dat er eerst nul of meer witte pixels komen, dan 3 zwarte, dan één of meer witte, dan 1 zwarte en tot slot nul of meer witte. De puzzel bestaat uit het vinden van een plaatje dat aan alle beschrijvingen voldoet. Een goede puzzel heeft precies één oplossing. Zie Wikipedia voor meer informatie. We nemen aan dat de hoogte en breedte maximaal 50 zijn.

In ons programma hebben we steeds de zogeheten huidige totaal-beschrijving en het huidige beeld. In het begin zijn hier alle lijnbeschrijvingen 0 en alle pixels wit. Verder zien we steeds het "huidige pixel", ook wel de "cursor" genoemd, in het begin ongeveer in het midden van het rooster. (Zorg ervoor dat op de een of andere manier deze locatie duidelijk gemarkeerd is.) De beschrijvingen staan rechts naast de rijen en onder de kolommen (dit is makkelijker te doen dan er voor en er boven, zoals meer gebruikelijk). Als je wilt kun je nog extra karakters definiëren (een punt bijvoorbeeld) voor pixels die, tijdens het puzzelen, zeker wit moeten zijn.

De gebruiker kan nu een aantal zaken doen:

1. Stoppen.
2. Een klein submenu ingaan, waarin:
   - de grootte van de puzzel kan worden gewijzigd, waarbij beschrijvingen en pixels weer 0 worden;
   - de gebruiker de symbolen die voor witte en zwarte pixels gebruikt worden kan kiezen (uiteraard twee verschillende);
   - de gebruiker kan instellen of bij verplaatsen van de ""cursor" het nieuwe punt wit of zwart wordt, of niet verandert;
   - het random-percentage (zie verderop) kan worden gewijzigd, tussen 0 en 100 procent.
3. Maak het huidige beeld leeg = schoon.
4. Random vullen van het huidige beeld, met (ongeveer) het door de gebruiker gekozen random-percentage zwarte pixels. Gebruik de random-generator van sheet 10 van het achtste college.
   ==========
5. De gebruiker kan de "cursor" één positie omhoog, omlaag, naar links of naar rechts bewegen, uiteraard binnen het rooster. Hierna wordt er opnieuw afgebeeld; het plaatje scrollt dus steeds omhoog. Gebruik speciale symbolen voor de plek van de cursor, op zowel een wit als een zwart pixel.
6. Toggelen: het huidige pixel wordt omgeklapt: 0 wordt 1, 1 wordt 0 (of preciezer: `false` wordt `true`, `true` wordt `false`).
7. De beschrijvingen worden de beschrijvingen van het huidige beeld. Het beeld klopt dan dus precies met de beschrijvingen.
   ==========
8. Inlezen van de huidige beschrijving uit een file. Formaat: de eerste regel bevat hoogte *m* en breedte *n*, gescheiden door een spatie. Daarna komen *m* regels met rij-beschrijvingen en *n* regels met kolom-beschrijvingen. Elke beschrijving wordt afgesloten met een 0; zo wordt `3 1 1` in de file `3 1 0` (met een spatie tussen de getallen). De beschrijving 0 wordt als 0 gerepresenteerd. Aangenomen mag worden dat de files wel het goede formaat hebben. Zie hier een (lastig) voorbeeld en nog een en nog een.
   Tip: lees getallen gewoon in met `invoer >> getal;`. Controleer ook of de file bestaat.
9. Wegschrijven van de huidige beschrijving naar een file.
10. [OPTIONEEL] Inlezen van het huidige beeld uit een file. Formaat: de eerste regel bevat hoogte *m* en breedte *n*, gescheiden door een spatie. Daarna *m* regels met een rij van het plaatje, met een 1 voor zwart en een 0 voor wit. Zie hier een voorbeeld.
    Wil je van een willekeurig plaatje `plaatje.jpg` een bestand in (bijna) dit formaat maken, doe dan het volgende (in Linux):
    ```
    convert plaatje.jpg -resize 40x40 plaatje.pbm
    pnmtoplainpnm plaatje.pbm > invoer.txt
    ```
    Uiteraard kan de gebruiker de filenaam kiezen. Als deze niet bestaat: een foutmelding, maar het programma komt weer in het menu terecht. Aangenomen mag worden dat de files wel het goede formaat hebben.
11. [OPTIONEEL] Wegschrijven van het huidige beeld.

Steeds staat bij correcte lijnen een "V": het gaat dus om rijen (ernaast) en kolommen (eronder) waarbij het huidige beeld precies aan de huidige beschrijving voldoet; dit verandert wellicht als de gebruiker iets aan het huidige beeld wijzigt. Als dit onderdeel geheel ontbreekt, kost dat 1 punt.

Als er in de beschrijvingen getallen met twee of meer cijfers staan, is dit lastig bij de kolommen. Druk dan bijvoorbeeld 10 als A af, 11 als B, enzovoorts.

De bedoeling is een klasse (`class`) `nonogram` te maken, met daarin onder meer functies die ieder voor zich een menuoptie afhandelen, zoals `vulrandom`. De parameters zijn typisch membervariabelen. Gebruik nog geen eigen headerfiles, alles moet deze keer in één file staan.

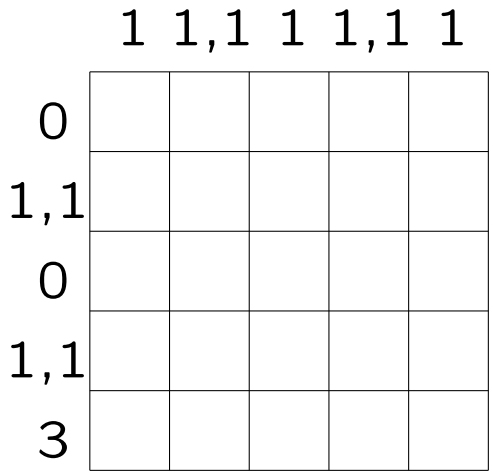
Opmerkingen
Gebruik geschikte (member)functies. Bij deze opgave mogen bij elke functie (zelfs `main`) tussen begin-{ en eind-} *hooguit circa 30* niet al te volle regels staan! Elke functie dient van commentaar voorzien te zijn, bij voorkeur één regel boven de functie. Let op goed parametergebruik: alle parameters, met uitzondering van membervariabelen, in de heading doorgeven, en de variabele-declaraties zowel bij `main` als bij de andere functies aan het begin. De enige te gebruiken headerfiles zijn in principe `iostream`, `fstream`, `cstdlib` en `string`. Zeer ruwe indicatie voor de lengte van het C++-programma: 500 regels. Denk aan het infoblokje.

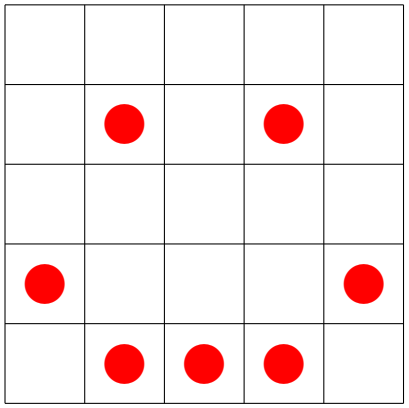
Uiterste inleverdatum: maandag 13 november 2023, 18:00 uur.

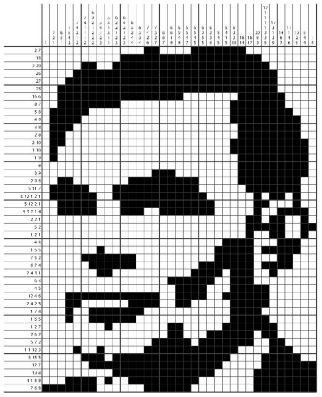
Manier van inleveren (één exemplaar per koppel, dat — ter herinnering — uit maximaal twee personen bestaat) is als volgt:

www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

Japanse puzzels (Nonogrammen) zien er zo uit:







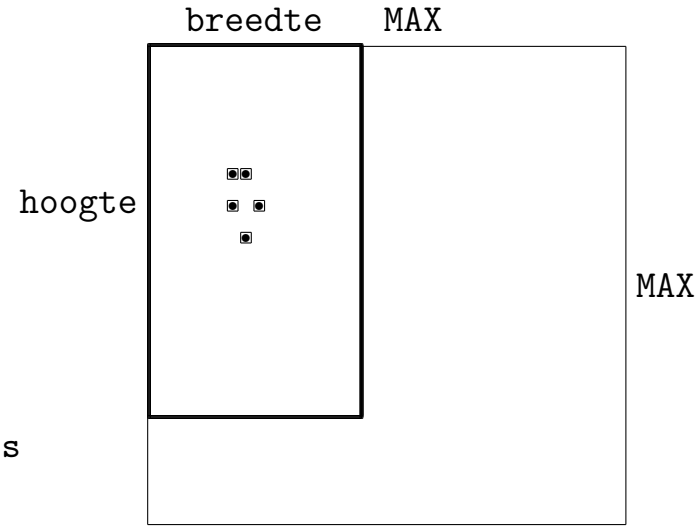
Naast iedere rij en boven (→ onder) iedere kolom staan in volgorde de lengtes van aaneengesloten series rode blokjes.

Voor Life/Nonogram/LightsOut: 2-dimensionale arrays (matrices)!

www.liacs.leidenuniv.nl/~kosterswa/pm/op3pm.php

Een klasse `nonogram` voor Nonogram ziet er ± zo uit:

```
class nonogram {
  public:
    nonogram ( );  // constructor
    void drukaf ( );
    void vulrandom ( );
    void maakschoon ( );
    void zetpercentage ( );
    // ...
  private:
    bool dewereld[MAX][MAX]; // array!!!
    int rijen[MAX][MAX];     // en nog zoiets
    int hoogte, breedte;
    int percentage;
    // ...
};//nonogram (let op de punt-komma hier)
```



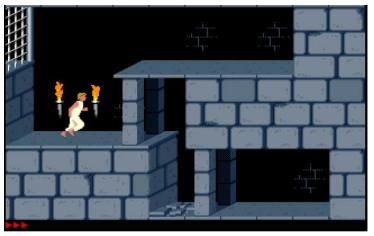
klasse object
↓ ↓

```
nonogram N;

N.drukaf ( );
```

Maak member-functies als (zie verderop):

```
// laat het nonogram zien
void nonogram::drukaf ( );
  ...
}//nonogram::drukaf
```



YouTube

en

```
// stel percentage in tussen 0 en 100
void nonogram::zetpercentage ( ) {
  percentage = leesGetal (100);
}//nonogram::zetpercentage
```

waarbij de zelfgemaakte functie `int leesGetal (int maxi)` een geheel getal, maximaal `maxi`, van toetsenbord inleest.

Voor een Nonogram-wereld is een 2-dimensionaal array nodig:

```
bool dewereld[MAX][MAX];
```

Er geldt: `dewereld[i][j]` is `true` precies dan als rij `i` (van boven) en kolom `j` (van links) "zwart" is.
En voor de rij-beschrijvingen:

```
int rijen[MAX][MAX];
```

En dit allemaal in een klasse `nonogram`, met methoden als `void nonogram::drukaf ( )`, zie eerder.

Maak eerst een *menu* en de functie `leesGetal`. Zie de tips:

www.liacs.leidenuniv.nl/~kosterswa/pm/pmwc7.php

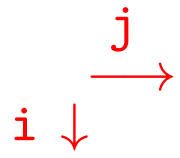
invoer moet $<$ 10000 zijn (bijvoorbeeld):

`\n\nabc123$%@@rr45xx\n` $\longrightarrow$ 1234 OF . . .

Enter

Een basisfunctie is dus het afdrukken van een Nonogram-wereld:

```
// laat de nonogram-wereld zien; eerste poging
void nonogram::drukaf ( );
  int i, j; // voor rijen en kolommen
  for ( i = 0; i < hoogte; i++ ) {
    for ( j = 0; j < breedte; j++ ) {
      if ( dewereld[i][j] )
        cout << " X"; // <== later "zwartkarakter"
      else            //     en de "cursor" apart
        cout << " .";
    }//for j
    cout << endl;
  }//for i
}//nonogram::drukaf
```



En is linksboven $(0, 0)$?

We willen willekeurige (random) getallen maken. Dit gaat met het volgende recept, de lineaire congruentie methode (zie Knuth). Kies een startwaarde x (de "seed"). Pas dan herhaald toe

$$x \leftarrow (a \cdot x + c) \text{ modulo } m$$

met vaste a, c en m; en voor modulo lees: % uit C++.

Vaak: $c = 1$, m een macht van 10, en a modulo $200 = 21$ (zie dictaat). Dan krijg je zoveel mogelijk verschillende getallen, voordat het zich gaat herhalen.

Met $x = (5 * x + 1) \,\%\, 8$ krijg je 0–1–6–7–4–5–2–3–0–...
En $x = (3 * x + 1) \,\%\, 8$ geeft 0–1–4–5–0–...

```
// geef random getal tussen 0 en 999
int randomgetal ( ) {
  static int getal = 42;                    // (*)
  getal = ( 221 * getal + 1 ) % 1000;  // niet aan knoeien
  return getal;
}//randomgetal
```

(*) Een `static` variabele is locaal, wordt eenmalig geinitialiseerd, en blijft behouden tussen functie-aanroepen.

En een random getal uit $\{1, 2, 3, 4, 5, 6\}$ aanmaken? Doe:

```
  cout << 1 + randomgetal ( ) % 6 << endl;
```

Of misschien beter `1 + randomgetal ( ) / 167` ?

In `cstdlib` zit de RNG ("(pseudo)Random Number Generator") `rand ( )`, en `srand (...)` zet de seed.

Met `int A[8];` maak je een array met 8 gehele getallen:
`A[0]`, `A[1]`, `A[2]`, `A[3]`, `A[4]`, `A[5]`, `A[6]`, `A[7]`.

↑ array-index

array-element

We vullen het array:

```
A[0] = 1;
for ( int i = 1; i < 8; i++ ) A[i] = 2 * A[i-1];
```

En we drukken het af:

```
for ( int i = 0; i < 8; i++ ) cout << A[i] << " ";
```

Dat geeft `1 2 4 8 16 32 64 128 .` Op `A[5]` staat 32.

Eendimensionale arrays

```
const int MAX = 1000;
int A[MAX]; // of int A[1000];
// declareert/definieert een array bestaande uit
// MAX integers A[0], A[1], ..., A[MAX-1]
```

en for-loop

```
for ( int i = 0; i < MAX; i++ )
  A[i] = 0;
// zet alle array-elementen op 0
```

Er is een subtiel verschil tussen *declareren* en *definiëren*.

... en functies

beginadres grootte

↓ ↓

```
void kwadraat (int B[ ], int n) { // geen & nodig!!
  for ( int i = 0; i < n; i++ )
    B[i] = i * i;
} // kwadraat

// vult de array-elementen B[0] tot en met B[n-1]
// met de eerste n kwadraten: 0 tot en met (n-1)^2
```

met aanroep: `kwadraat (A,MAX);` of `kwadraat (A,500);`. Array `A` verandert, ook al is het een call-by-value parameter!

Hoe sorteer je een array oplopend? Een eerste idee is: zet herhaald de "kleinste" vooraan.

```
void simpelsort (int C[ ], int n) {
  int voorste, kleinste, plaatskleinste, k;
  for ( voorste = 0; voorste < n; voorste++ ) {
    plaatskleinste = voorste;
    kleinste = C[voorste];
    for ( k = voorste + 1; k < n; k++ )
      if ( C[k] < kleinste ) {
        kleinste = C[k];
        plaatskleinste = k;
      }//if
    if ( plaatskleinste > voorste )
      wissel (C[plaatskleinste],C[voorste]);
  }//for
}//simpelsort
```

zoek (plaats)kleinste van C[voorste],...,C[n-1]

Een voorbeeld van de werking van simpelsort:

```
0  1  2  3  4  5  6  (n=7)

3  8  7  5  2  4  9
2| 8  7  5  3  4  9
2  3| 7  5  8  4  9
2  3  4| 5  8  7  9
2  3  4  5| 8  7  9
2  3  4  5  7| 8  9
2  3  4  5  7  8| 9
2  3  4  5  7  8  9|
```

Bubblesort

En nog een (slechte) sorteermethode:

```
void bubblesort (int A[ ], int n) {
  int i, j;
  for ( i = 1; i < n; i++ )
    for ( j = 0; j < n - i; j++ )
      if ( A[j] > A[j+1] )
        wissel (A[j],A[j+1]);  // (*)
}//bubblesort
```

YouTube

Bij (*):

```
void wissel (int & a, int & b) {
  int hulp = a; a = b; b = hulp; }//wissel
```

of (zonder functie `wissel`):

```
  { int temp = A[j]; A[j] = A[j+1]; A[j+1] = temp; }
```

Een voorbeeld van de werking van simpelsort (links) en bubblesort (rechts):

```
0  1  2  3  4  5  6  (n=7)      0  1  2  3  4  5  6

3  8  7  5  2  4  9              3  8  7  5  2  4  9
2| 8  7  5  3  4  9              3  7  5  2  4  8 |9
2  3| 7  5  8  4  9              3  5  2  4  7 |8  9
2  3  4| 5  8  7  9              3  2  4  5 |7  8  9
2  3  4  5| 8  7  9              2  3  4 |5  7  8  9
2  3  4  5  7| 8  9              2  3 |4  5  7  8  9
2  3  4  5  7  8| 9              2 |3  4  5  7  8  9
2  3  4  5  7  8  9|
```

Bubblesort doet bij een rij met n elementen

$$(n-1)+(n-2)+\ldots+3+2+1 = n(n-1)/2$$

vergelijkingen tussen array-elementen. Het is een $O(n^2)$ ("orde n^2") algoritme — en dat is niet zo fijn.

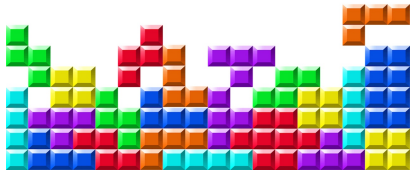
Dezelfde analyse geldt voor "simpelsort" = Selection sort.

Later meer over zoeken en sorteren ... het kan namelijk beter = sneller!

http://www.sorting-algorithms.com/

Insertion sort (invoegsorteer)

```
// sorteer array A (met n integers) oplopend
// met behulp van insertion sort (Opgave 54)
void invoegsorteer (int A[ ], int n) {
  int i, j, temp;
  for ( i = 1; i < n; i++ ) {  // zet A[i] goed in
    temp = A[i];      // reeds gesorteerd beginstuk
    j = i - 1;
    while ( ( j >= 0 ) && ( A[j] > temp ) ) {
      A[j+1] = A[j];
      j--;
    }//while
    A[j+1] = temp;
  }//for
}//invoegsorteer
```



link

Insertion sort: werking

3	8	14	17	32	33	37	16	12	58	7	1	34	6	9

0 1 2 ... $i-1$ i ... $n-1$

okee

In de i^{de} ronde is
$A[0]$ tot en met $A[i-1]$
van array A (met n elementen)
al gesorteerd en wordt $A[i]$
in het beginstuk op de juiste
plek "ingevoegd".

Het aantal vergelijkingen tussen array-elementen dat dit sorteeralgoritme doet hangt af van het invoerrijtje (met n elementen).

In het slechtste geval (worst case) kost het

$$1 + 2 + 3 + \ldots + i - 1 + \ldots + n - 1 = \frac{1}{2}n(n-1)$$

vergelijkingen om het array oplopend te sorteren, bijvoorbeeld als het beginarray aflopend gesorteerd is.

Het beste geval (best case) treedt op als het beginarray reeds oplopend gesorteerd is: $n - 1$ vergelijkingen.

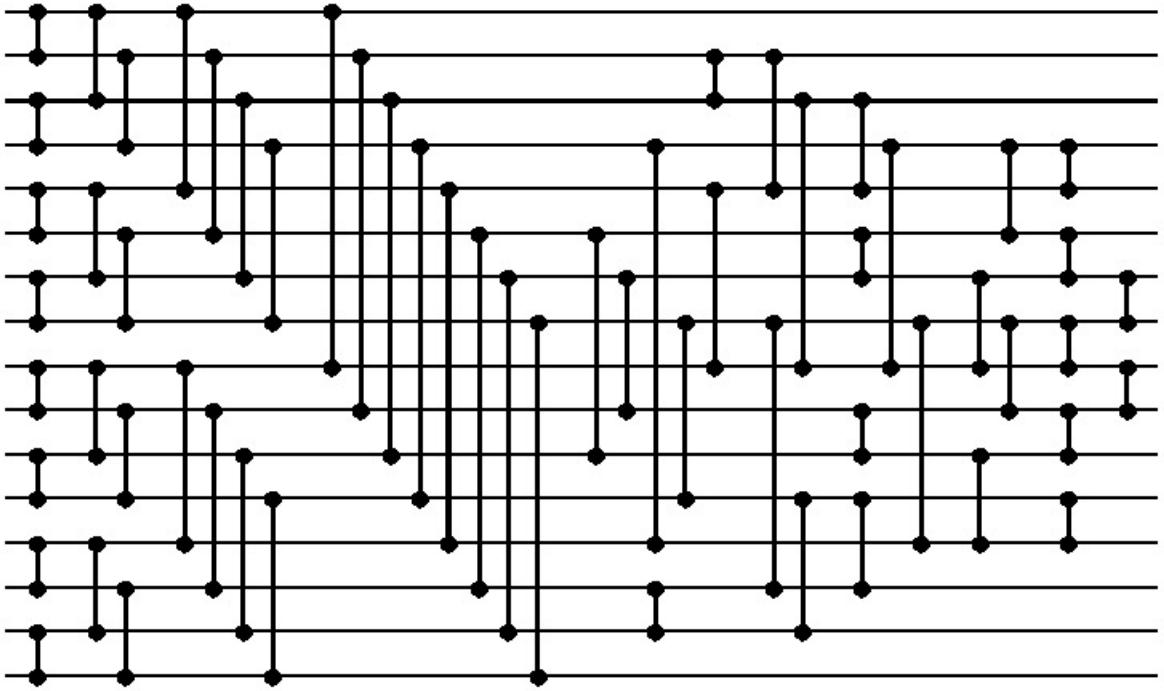
Shellsort ⋈

↑

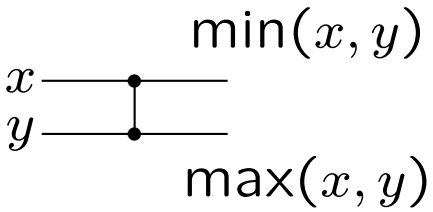
geen tentamenstof

```
void Shellsort (int A[ ], int n) {
  int i, j, h = n; // spronggrootte h
  while ( h > 1 ) {
    // A is h-gesorteerd
    h = h / 2;  // er bestaan betere keuzes dan / 2
    for ( i = h; i < n; i++ ) {
      // insertion sort op deelrijtjes
      temp = A[i]; j = i - h;
      while ( ( j >= 0 ) && ( A[j] > temp ) ) {
         A[j+h] = A[j];
         j = j - h;
      }//while
      A[j+h] = temp;
    }//for
  }//while
  // h = 1: A is nu 1-gesorteerd = gesorteerd
}//Shellsort
```

Er zijn allerlei andere benaderingen, zoals “counting sort” (voor 1323312132312 3), en sorteernetwerken (GPU’s):



vergelijker:



sorteert
16 getallen
60 vergelijkers
10 tijdstappen

sorteermethode	aantal vergelijkingen (worst case)
Selection sort	$O(n^2)$
Bubblesort	$O(n^2)$
Insertion sort	$O(n^2)$
Shellsort	$O(n\sqrt{n})$ (of nog beter?)
Quicksort	$O(n \lg n)$

Deze sorteeralgoritmen zijn alle gebaseerd op het doen van array-vergelijkingen (Selection sort = simpelsort); $\lg n = {}^2\log n = \log_2 n$.

Stelling: Elk sorteeralgoritme gebaseerd op het doen van array-vergelijkingen doet in het slechtste geval (de worst case) altijd *minstens* $\lg n! \approx c\, n \lg n$ vergelijkingen voor een array met n elementen.

zie de vakken Algoritmiek en Complexiteit . . .

```
void tabelletje (int n) {
  int i, j;
  for ( i = 1; i <= n; i++ ) { // buitenste loop
    cout << i << ": ";
    for ( j = 1; j <= i; j++ ) // binnenste loop
      cout << i * j << " ";
    cout << endl;
  }//for i
}//tabelletje
```

geeft, met `tabelletje (5);`:

```
1: 1
2: 2 4
3: 3 6 9
4: 4 8 12 16
5: 5 10 15 20 25
```

Tweedimensionale arrays (2D arrays, matrices)

```
const int m = 100;     // rijen
const int n = 50;      // kolommen
int A[m][n];           // of int A[100][50];
// declareert een tweedimensionaal array van m rijen
// en n kolommen, bestaande uit m*n integers
```

en dubbele for-loop

```
int i, j;
for ( i = 0; i < m; i++ )
  for ( j = 0; j < n; j++ )
    A[i][j] = 42;      // dus niet A[i,j]
// zet alle array-elementen op 42
```

... en functies

moet constante zijn!!
`B[ ][n]` mag ook (als `n const` is)

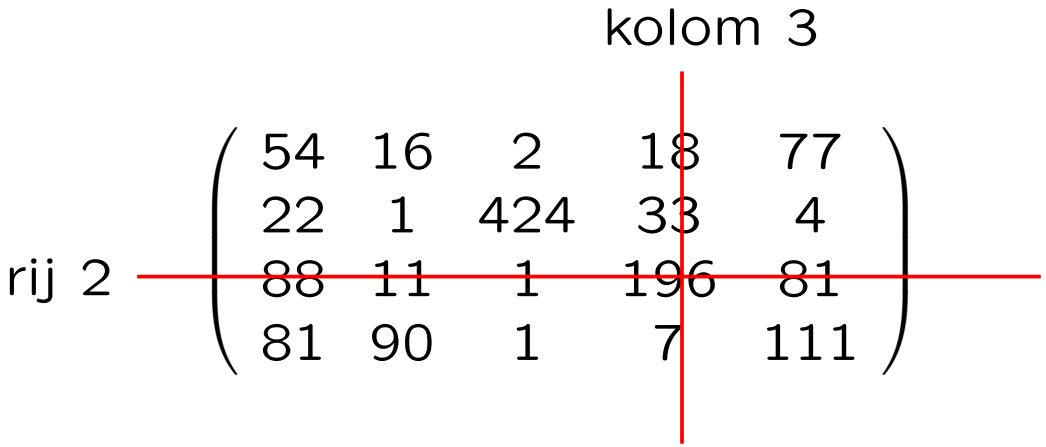
↓

```
int somarray (int B[ ][50], int zoveel) {
  int i, j, som = 0;
  for ( i = 0; i < zoveel; i++ )   // rijen
    for ( j = 0; j < 50; j++ )     // kolommen
      som += B[i][j];   // += betekent "ophogen met"
  return som;
}//somarray

// berekent de som van de elementen
// uit de eerste zoveel rijen van B
```

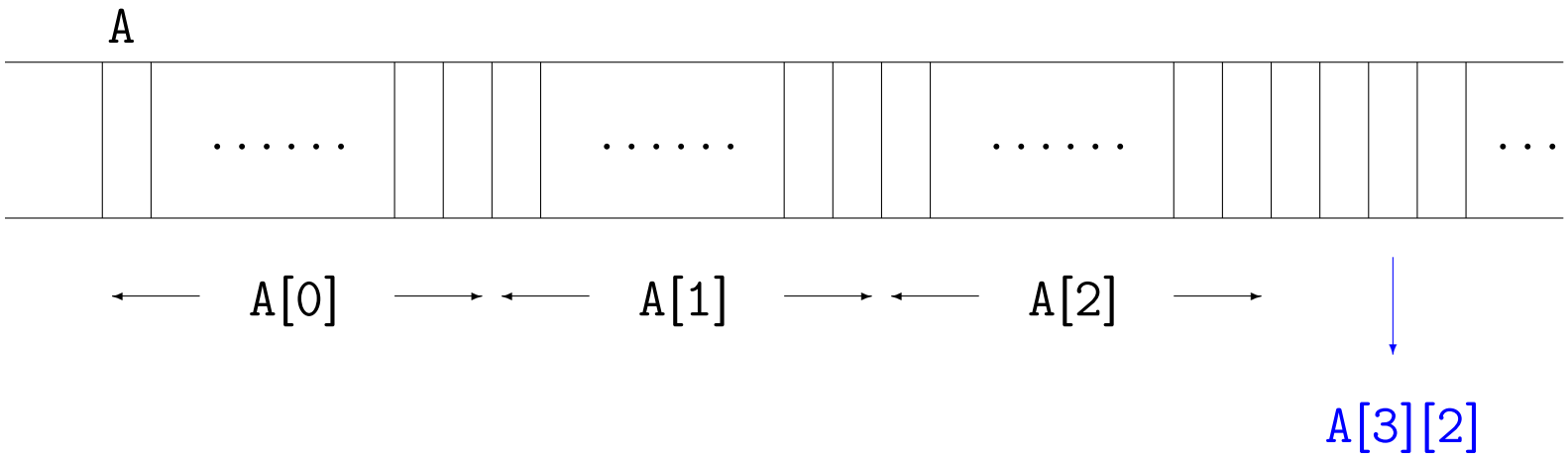
en aanroep: `antwoord = somarray (A,m);` of
`antwoord = somarray (A,20);` of ...

Een 4×5 array `A` (`int A[4][5];`) heeft 4 rijen en 5 kolommen:

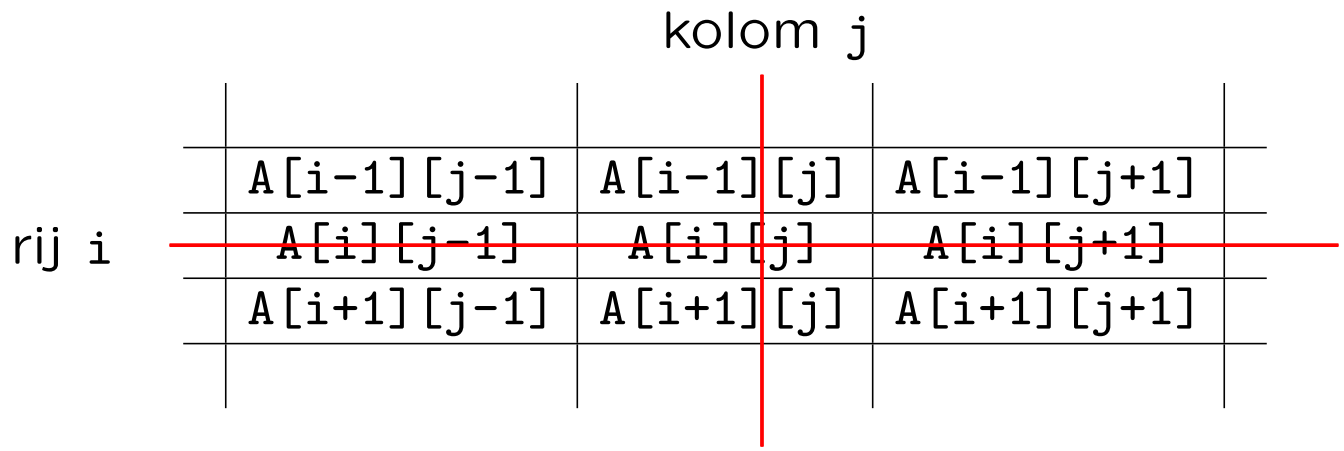


Op plek $(2, 3)$ staat het getal 196: `A[2][3]`.
Op plek $(0, 0)$ (dus linksboven) staat het getal 54: `A[0][0]`.
Rechtsonder staat `A[3][4]`: 111.
En het array-element `A[4][5]` "bestaat niet".

De rijen van zo'n 2-dimensionaal array `int A[m][n]` liggen achter elkaar in het geheugen:



Het adres van `A[3][2]` is `A + 3 * n + 2`, of eigenlijk preciezer `A + (3 * n + 2) * sizeof (int)`.
De `n` moet dus bekend, oftewel een `const`, zijn!



`m` × `n` array `int A[m][n];`

Array-elementen aan een rand hebben minder buren!

Als een functie de inhoud van het array moet veranderen is geen & nodig:

adres verandert niet ↓ dus ↓

```
void ikeerj (int A[ ][n], int m) {   // geen & !!!
  int i, j;
  for ( i = 0; i < m; i++ )
    for ( j = 0; j < n; j++ )
      A[i][j] = i * j;
}//ikeerj
```

↑ inhoud verandert wel

We bekijken nu `n` bij `n` vierkante matrices:

```
const int n = 42;

// C wordt de som van A en B, alle drie n bij n matrices
// optelling geschiedt elementsgewijs
void optellen (double A[ ][n], double B[ ][n],
               double C[ ][n]) {
  int i, j;
  for ( i = 0; i < n; i++ )
    for ( j = 0; j < n; j++ )
      C[i][j] = A[i][j] + B[i][j];
}//optellen
```

zie het vak Lineaire algebra . . .

We bekijken weer `n` bij `n` vierkante matrices:

```
const int n = 42;

// C wordt het (matrix-)product van A en B
void vermenigvuldigen (double A[ ][n], double B[ ][n],
                       double C[ ][n]) {
  int i, j, k;
  for ( i = 0; i < n; i++ )
    for ( j = 0; j < n; j++ ) {
      C[i][j] = 0;
      for ( k = 0; k < n; k++ )
        C[i][j] += A[i][k] * B[k][j];
    }//for j
}//vermenigvuldigen
```

Voorbeeld (we berekenen van $A \cdot B = C$ met name `C[1][0]`):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

In het algemeen geldt:

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] * B[k][j] \text{ ofwel } C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj}$$

Het product van twee matrices is overigens ook gedefinieerd voor niet-vierkante matrices A en B, mits maar geldt dat aantal kolommen van A = aantal rijen van B.

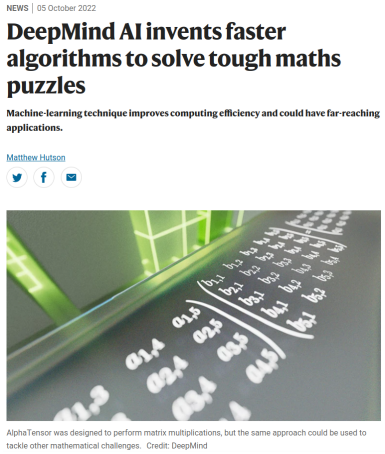
Matrixvermenigvuldiging: Strassen

Het gewone algoritme (links) kost $O(n^3)$ vermenigvuldigingen van array-elementen voor het product van twee $n \times n$ matrices; Strassen (rechts) "slechts" $O(n^{\log_2 7}) \approx O(n^{2.8})$:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$\begin{aligned} 19 &= 1*5+2*7 \\ 22 &= 1*6+2*8 \\ 43 &= 3*5+4*7 \\ 50 &= 3*6+4*8 \end{aligned}$$

$$\begin{aligned} M_1 &= (1+4)*(5+8) = 65 \\ M_2 &= (3+4)*5 = 35 \\ M_3 &= 1*(6-8) = -2 \\ M_4 &= 4*(7-5) = 8 \\ M_5 &= (1+2)*8 = 24 \\ M_6 &= (3-1)*(5+6) = 22 \\ M_7 &= (2-4)*(7+8) = -30 \\ 19 &= M_1+M_4-M_5+M_7 \\ 22 &= M_3+M_5 \\ 43 &= M_2+M_4 \\ 50 &= M_1-M_2+M_3+M_6 \end{aligned}$$



NEWS | 05 October 2022

DeepMind AI invents faster algorithms to solve tough maths puzzles

Machine-learning technique improves computing efficiency and could have far-reaching applications.

Matthew Hutson

AlphaTensor was designed to perform matrix multiplications, but the same approach could be used to tackle other mathematical challenges. Credit: DeepMind

Opgave 1 van het tentamen van 6 januari 2014:

In een array `int A[n]` staan `n` (een `const` > 0) gehele getallen.

a. Schrijf een C++-functie `hoevaak (A,X,n)` die teruggeeft hoe vaak het gehele getal `X` in het array `A` voorkomt.

b. Schrijf een Booleaanse C++-functie `uniek (A,n)` die precies dan `true` teruggeeft als geen enkel getal twee maal (of vaker) voorkomt in `A`, en anders `false`. Hierbij moet de functie van **a** *zinvol* gebruikt worden (hoe vaak komt `A[i]` voor?).

c. Schrijf een C++-functie `meest (A,n)` die het meest voorkomende getal uit `A` teruggeeft. Als er verschillende kandidaten zijn (bijvoorbeeld voor het array `17 12 30 12 42 30`) moet het kleinste getal dat het meest voorkomt worden geretourneerd. In het voorbeeld is dit `12` (dat even vaak voorkomt als `30`). Maak opnieuw gebruik van de functie van **a**.

d. Schrijf een C++-functie `sorteer (A,n)` die de getallen in `A` zodanig ordent dat voor alle getallen (behalve het laatste) geldt dat ze hooguit even vaak voorkomen als hun rechter buurman. Tip: pas de C++-code voor *bubblesort* eenvoudig aan; gebruik **a**.

e. Hoe vaak wordt de functie `hoevaak` aangeroepen in **d**?

zie werkcollege 2–3 november: 1a en 2a

- werk aan de derde programmeeropgave — de deadline is op maandag 13 november 2023
- bezoek daarom de colleges (video's), werkcolleges en vragenuren
- lees Savitch Hoofdstuk 5
- lees collegedictaat Hoofdstuk 3.8, 4.1 en 4.2
- maak opgaven 31/36 uit het opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/