IFG1 - Übungen 1
Algorithmierung

Pseudocode / Struktogramme / PAP

HTWK - 18STB

ab 27.Mai 2020

Aufg.1a: Skalarprodukt

Formulieren sie ein Struktogramm für die Berechnung des Skalarproduktes zweier n-dimensionaler Vektoren

geg: Vektor $a = (a_1, a_2,...,a_n)^T$
Vektor $b = (b_1, b_2,...,b_n)^T$

$$s = a^T b = |a| \cdot |b| \cdot \cos \sphericalangle(a,b)$$

$$s = \sum_{i=1}^{n} a_i \cdot b_i$$

ges: Skalarprodukt $s=a^Tb$ zwischen den beiden Vektoren a und b.

IN: n, a_i i=1(1)n, b_i i=1(1)n
S=0;
FOR (i=1; i<=n; i++) DO { S+= $a_i \cdot b_i$; }
OUT: "...", S

$S = S + a_i \cdot b_i$;

Aufg.1b: dyadisches Produkt

ges: dyadisches Produkt
(Matrix c=c(i,j) mit i=1(1)n, j=1(1)m)
zwischen den beiden Vektoren a und b.

Def: Dyadisches Produkt ab^T zweier Vektoren
$(a_1,...,a_n)^T$ und $(b_1,...,b_m)^T$

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \begin{pmatrix} b_1 & \dots & b_m \end{pmatrix} = \begin{pmatrix} a_1 \cdot b_1 & \dots & a_1 \cdot b_m \\ \vdots & & \vdots \\ a_n \cdot b_1 & \dots & a_n \cdot b_m \end{pmatrix}$$

$$C_{ij} = a_i \cdot b_j$$

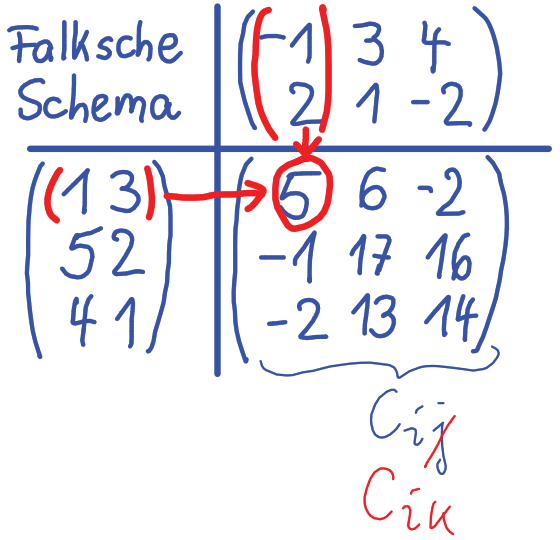
IN: $n, a_i \quad i=1(1)n$
$m, b_j \quad j=1(1)m$

FOR $(i=1; i \le n; i++)$ DO
FOR $(j=1; j \le m; j++)$ DO
$\{ C_{ij} = a_i \cdot b_j \}$

OUT: "dyad. Produkt!",
$C_{ij} \quad i=1(1)n, j=1(1)m$

Aufg.1c: Matrizenprodukt

geg: Matrizen A(n,m), B(m,l)
ges: Matrixprodukt C(n,l)=A(n,m)xB(m,l)



IN: n, m, l, a_{ij} $i=1(1)n, j=1(1)m$
b_{jk} $j=1(1)m, k=1(1)l$

```
FOR (i=1; i≤n; i++) DO
  FOR (k=1; k≤l; k++) DO
    // berechne Skalarprodukt aus Zeile i von A und Spalte k von B
    S=0;   <- auch c_ik=0 aber uneffizient
    FOR (j=1; j≤m; j++) DO { s += a_ij · b_jk }
    c_ik = s;
```

skalare Variable → S

c_{ik}

OUT: c_{ik} $i=1(1)n, k=1(1)l$

Aufg.2a: Distanz und Spannweite

geg: Vektor a(i) i=1(1)n mit a(i)=reelle Zahl

	1	2	3	4	5	6	7	8
z.B. a=	1.3	23.4	-12.6	5.8	9.3	-3.1	9.4	-4.5

ges: Distanz = maximales Element - minimales Element

Spannweite = Index des maximalen Elementes
- Index des minimalen Elementes

Notieren Sie einen Algorithmus zur Bestimmung von Distanz und Spannweite in einem Vektor von Zahlen! Berücksichtigen Sie auch eventuelle Mehrdeutigkeiten!

Aufg.3a: Algorithmen mit 01-Vektoren (maximale Differenz zweier Einsen)

geg: Vektor a(i) i=1(1)n mit a(i)=0 oder 1

```
                1 2 3 4 5 6 7 8 9 10 11 12
      z.B. a=(0,0,1,0,1,1,0,1,0,0, 0, 1)
```

ges: maximale Distanz d zweier Einsen in diesem Vektor, d.h. die maximale Differenz von Indizes, an deren Stellen in diesem Vektor Einsen stehen.

$$d = \max \{(j-i) \mid a(j)=1 , a(i)=1 \}$$

Falls nur ein Vektorelement gleich 1 ist, so sei d=0 und falls keine 1 im Vektor auftritt so sei d=n. (im Bsp. ist d=12-3=9)

Aufg.3b: Algorithmen mit 01-Vektoren (Wechsel von 0 auf 1 und von 1 auf 0)

geg: Vektor a(i) i=1(1)n mit a(i)=0 oder 1

```
                1 2 3 4 5 6 7 8 9 10 11 12
   z.B. a=(0,0,1,0,1,1,0,1,0,0, 0, 1)
```

ges: Anzahl z1 der Wechsel in dem Vektor von Wert 0 auf den Wert 1 und die Anzahl z2 der Wechsel in dem Vektor von Wert 1 auf den Wert 0.
(im Bsp. ist z1=4 und z2=3)

$0 \to 1 \quad a_{i+1} - a_i = 1$

$1 \to 0 \quad = -1$

$0 \to 0 \quad = 0$

$1 \to 1 \quad = 0$

```
IN: n, a_i  i=1(1)n
z1=0; z2=0;  i<n
FOR (i=1; i<=n-1; i++) DO
   IF ((a_{i+1} - a_i) == 1) THEN z1++
   ELSE IF ((a_{i+1} - a_i) == -1) THEN z2++;
        ELSE;
OUT: z1, z2
```

2 mgl. Indexberech.

d = a_{i+1} - a_i; (d: Operation)

Aufg.3c: Algorithmen mit 01-Vektoren (Differenz der Anzahl von 0en und 1en)

geg: Vektor a(i) i=1(1)n mit a(i)=0 oder 1

```
                1 2 3 4 5 6 7 8 9 10 11 12
     z.B. a=(0,0,1,0,1,1,0,1,0,0, 0, 1)
```

ges: Differenz diff der Anzahlen von Einsen und Nullen in diesem Vektor.
(im Bsp. ist diff=5-7= -2)

z1 z0
= n - z1 ⇒ diff = z1 - z0
= z1 - (n - z1)
= 2z1 - n

diff = z1 - z0
alternativ
diff = 2*z1 - n // denn z0 = n - z1 (und muß demzufolge nicht berechnet werden)
A: "....", diff

Aufg.3d: Algorithmen mit 01-Vektoren (Vektor der Lauflängen - RLE)

RLE = Run Length Encoding

geg: Vektor a(i) i=1(1)n mit a(i)=0 oder 1

```
                1 2 3 4 5 6 7 8 9 10 11 12
   z.B. a=(0,0,1,0,1,1,0,1,0,0, 0, 1)
```

Sonderfälle!

ges: Folge der Lauflängen, d.h. die Anzahlen gleicher aufeinanderfolgender Stellen von Nullen und Einsen (beginnend mit der Lauflänge von Nullen, und falls der Vektor mit einer Eins beginnt, dann wird die Nullen Lauflänge =0 vorangestellt)

z.B. a=(0,0,1,1,1,1,0,1,1,0,0, 0, 1) ➔ (2,4,1,2,3,1) = Lauflängenvektor = LL

2

Nullen-Lauflänge nach Def.

IN: n, a_i i=1(1)n //Eingabeverifizierung

{IF (a_1=1) THEN LL_1=0; l=1; k=2;
ELSE l=1; k=1;

{l=0; k=1;
IF (a_1=1) THEN { LL_k=0; l=1 }

Zähler für gleiche Bits

FOR (i=2; i≤n; i++) DO
IF ($a_i \neq a_{i-1}$) THEN {k++; LL_k=l; l=1}
ELSE l++;
k++; LL_k=l;
OUT: "Lauflängenvektor", LL_j j=1(1)k

Aufg.4a: Algorithmen mit mehreren 01-Vektoren (Stellendistanz / Hammingabstand)

geg: Vektoren a(i) i=1(1)n mit a(i)=0 oder 1 , b(i) i=1(1)n mit b(i)=0 oder 1

```
                1 2 3 4 5 6 7 8 9 10 11 12
z.B.  a=(0,0,1,0,1,1,0,1,0,0, 0, 1)
      b=(1,0,1,1,0,1,0,1,0,1, 0, 0)
```

sd = 5

ges: Hammingabstand/Stellendistanz zwischen den beiden Codevektoren, d.h. die Anzahl h der Stellen i in denen sich die Vektoren a und b unterscheiden,
d.h. die Anzahl h der Stellen i in denen a(i)≠b(i).

IN: n, a_i, b_i i=1(1)n
sd = 0; // Initialisierung des Zählers

FOR (i=1, i≤n; i++) DO
  IF ($a_i \neq b_i$) THEN sd++;

OUT: sd

<> !=

Aufg.4b: Algorithmen mit mehreren 01-Vektoren (Stellendistanz / Hammingabstand)

Gegeben sei eine (mxn)-Matrix A mit a_{ij} =0 oder a_{ij} =1 für i=1(1)m, j=1(1)n, d.h.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

⇒ Stellendistanzmatrix

Diese 0,1-Matrix A kann als eine Code-Matrix aufgefasst werden mit m-Codevektoren (Zeilenvektoren).

ges: Algorithmus zur Bestimmung des Hammingabstandes des durch A definierten Codes, d.h. der minimalen Stellendistanz zwischen je zwei Codevektoren.

```
IN: m, n  a_ij mit i=1(1)m, j=1(1)n
// jede Zeile i mit jeder anderen Zeile ii vergleichen, i<ii
MIN = 1E30;
FOR (i=1; i≤m-1; i++)DO
  FOR (ii = i+1; ii≤m; ii++)DO
      // Stellendistanz zwischen Zeile i und Zeile ii berechnen
      dd = 0;
      FOR (j=1; j≤n; j++)DO
          IF (a_ij ≠ a_ii,j) THEN dd++;
      d_i,ii = dd; // nicht unbedingt notwendig, aber nützlich
      IF (dd < MIN) THEN {MIN = dd; I1MIN=i; I2MIN=ii;}
OUT: "Hammingabstand im Code =", MIN
```

Aufg.4c: Algorithmen mit mehreren 01-Vektoren (orthogonale Codes)

Gegeben sei eine (mxn)-Matrix A mit a_{ij} =0 oder a_{ij} =1 für i=1(1)m, j=1(1)n, d.h.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Diese 0,1-Matrix A kann als eine Code-Matrix aufgefasst werden mit m-Codevektoren (Zeilenvektoren).

ges: Gesucht ist ein Algorithmus, mit dem festgestellt werden kann, ob alle Codevektoren von A zueinander orthogonal sind, d.h. ob für alle Indexpaare (k,l) mit k≠l das Skalarprodukt zwischen der k-ten und l-ten-Zeile von A gleich Null ist.

Aufg.5a: Algorithmen mit Matrizen (Zeilen in Matrix duplizieren)

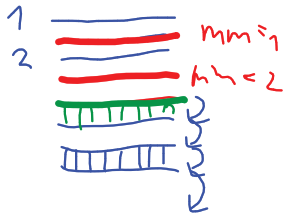
Gegeben sei eine (mxn)-Matrix A mit reellen Zahlen a_{ij} für i=1(1)m, j=1(1)n, d.h.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

ges: Gesucht ist ein Algorithmus, der **hinter** (unter) jeder Zeile dieser Matrix, in der die Anzahl der Einsen größer ist als die Anzahl der Nullen, eine neue Zeile mit genau den gleichen Werten, wie in der entsprechenden Vorgängerzeile einfügt (d.h. diese Zeilen nach hinten duplizieren und die restlichen Matrixzeilen jeweils nach hinten/unten verschieben).

```
IN: m,n, a_ij  i=1(1)m, j=1(1)n        mm=0
FOR (i=1; i≤m; i++)DO
  // Berechne d: Anz.Einsen - Anzahl Nullen in i-ten Zeile → HA!
  IF (d>0) THEN  K: alle Zeilen ab Zeile i nach unten "kopieren".
    FOR ( ii = m+mm; ii ≥ i+m; ii--) DO
      K: kopiere Zeile ii nach Zeile ii+1
      FOR (j=1; j≤n; j++)DO a_ii+1,j = a_ii,j
    mm = mm+1;   // mm++
```

i+m



→ alternativ DO...WHILE

5 6 duplizieren 9 10 11

Aufg.5b: Algorithmen mit Matrizen (Zeilen in Matrix löschen)

Gegeben sei eine (mxn)-Matrix A mit reellen Zahlen a_{ij} für i=1(1)m, j=1(1)n, d.h.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

ges: Gesucht ist ein Algorithmus zum Löschen aller derjenigen Zeilen der Matrix, in denen die Anzahl der Einsen kleiner als die Anzahl der Nullen ist (d.h. Zeilen löschen und Matrix verdichten).

Aufg.5c: Algorithmen mit Matrizen (Dreiecksungleichung in Entfernungsmatrix)

Gegeben sei eine (mxn)-Matrix A mit reellen Zahlen a_{ij} für i=1(1)m, j=1(1)n, d.h.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

→ siehe auch Warshall-Algorithm. Floyd-Algorithmus Bellmann-Ford-Algorith.

ges: Interpretieren Sie die Elemente a_{ij} mit j>i der Matrix A als Entfernungen von einem Ort i zu einem Ort j. Prüfen Sie ob für die Entfernungsdreiecksmatrix die Dreiecksungleichung erfüllt ist, d.h. ob für alle a_{ij} mit j>i gilt:

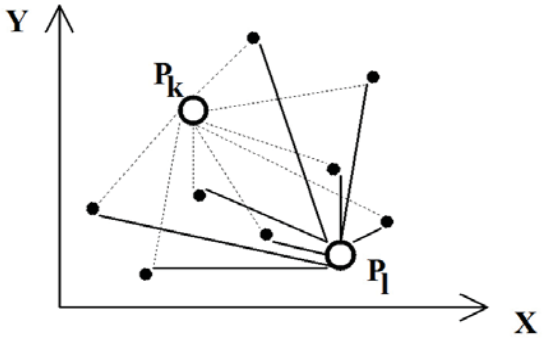
$a_{ij} \le a_{ki} + a_{kj}$ für alle k mit k<i
$a_{ij} \le a_{ik} + a_{kj}$ für alle k mit i<k<j
$a_{ij} \le a_{ik} + a_{jk}$ für alle k mit j<k .

ähnlich: Floyd-Algorith. Warshall-Algorithmus Bellman-Ford-Alg.

Geben Sie jene Indextripel i,j,k aus, für die eine Ungleichung nicht erfüllt ist, d.h. für die der Umweg über k kürzer als die direkte Verbindung ist.

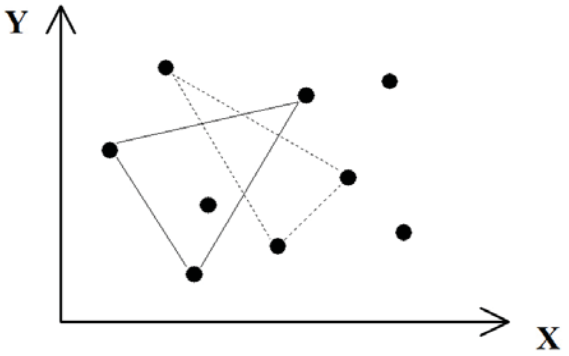
Aufg.6a: Abstands- und Distanzprobleme in der Ebene

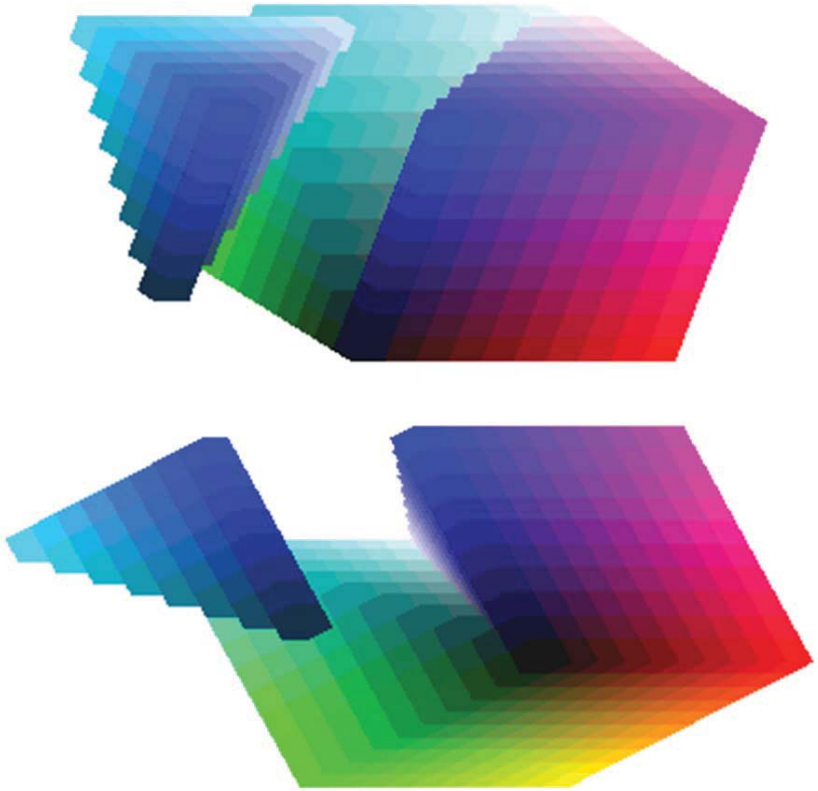
Entwerfen Sie einen **Algorithmus** zu dem folgenden Abstandssummenproblem:
Gegeben seien die Koordinaten (x_i,y_i) i=1(1)n von n Orten in der Ebene (idealisiert als Punkte).
Von diesen n Orten sollen zwei Orte P_k, P_l mit $P_k \neq P_l$ so ausgewählt werden, dass die beiden Abstandssummen
S1 (= Summe der Abstände von P_k nach allen anderen Punkten, außer nach P_l) und
S2 (= Summe der Abstände von P_l nach allen anderen Punkten, außer nach P_k)
möglichst wenig voneinander differieren.



Aufg.6b: Abstands- und Distanzprobleme in der Ebene

Entwerfen Sie einen **Algorithmus** zu dem folgenden Distanzierungsproblem: Gegeben seien die Koordinaten (x_i,y_i) i=1(1)n von n Orten in der Ebene (idealisiert als Punkte).
Drei Personen A,B,C sollen derart auf drei jeweils verschiedene Orte verteilt werden, dass die Summe der euklidischen Abstände (kürzeste Entfernungen) zwischen je zwei der Personen so groß wie möglich ausfällt.
Welche dieser Punkte maximieren die Summe der 3 Entfernungen?

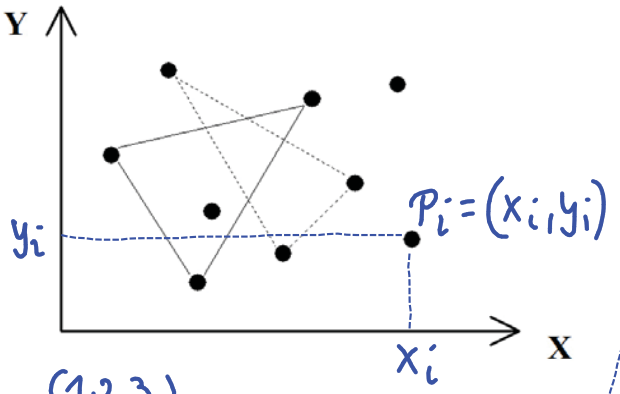




```
FOR (i=1;i<=n-2;i++) DO
  FOR (j=i+1;j<=n-1;j++) DO
      FOR (k=j+1;k<=n;k++) DO
```

Aufg.6c: Abstands- und Distanzprobleme in der Ebene

Entwerfen Sie einen **Algorithmus** zu dem folgenden Distanzierungsproblem:
Gegeben seien die Koordinaten (x_i,y_i) i=1(1)n von n Orten in der Ebene
(idealisiert als Punkte).
Drei Personen A,B,C sollen derart auf drei jeweils verschiedene Orte verteilt werden,
dass die Differenz zwischen der längsten und kürzesten Seite des entstehenden Dreieckes
so klein wie möglich ausfällt, d.h. das entstehende Dreieck so „gleichseitig“ wie möglich wird.
Welche dieser Punkte realisieren eine solche Minimal-Eigenschaft?



(1,2,3)
(1,2,4) (1,3,4)
(1,2,5) (1,3,5) (1,4,5)
⋮
(1,2,n) (1,3,n) (1,4,n)
⋮
(1,n-1,n) → (n-2; n-1; n)

IN: n, (x_i,y_i) mit i=1(1)n
MINDIFF = 1E30; P1=0; P2=0; P3=0;
FOR (i=1; i≤n-2; i++) DO //1.Vgl.punkt
FOR (j=i+1; j≤n-1; j++) DO //2.Vgl-p.
FOR (k=j+1; k≤n, k++) DO //3.VP
// P_i, P_j, P_k sind ausgewählt
$L1=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$ // $\overline{P_iP_j}$
$L2=\sqrt{(x_j-x_k)^2+(y_j-y_k)^2}$ // $\overline{P_jP_k}$
$L3=\sqrt{(x_i-x_k)^2+(y_i-y_k)^2}$ // $\overline{P_iP_k}$
// MIN = min(L1,L2,L3) ?
// MAX = max(L1,L2,L3) ?
D = MAX − MIN
IF (D < MINDIFF) THEN
{MINDIFF = D; P1=i; P2=j; P3=k;}
OUT: MINDIFF, P1, P2, P3

Aufg.6bb/cc: Abstands- und Distanzprobleme in der Ebene

Analog Aufg.6b, aber gesucht sind vier Punkte, so dass

- der Umfang des zwischen diesen 4 Punkten aufgespannten Rechtecks möglichst groß wird
- der Umfang des zwischen diesen 4 Punkten aufgespannten Rechtecks möglichst klein wird
- der Flächeninhalt des zwischen diesen 4 Punkten aufgespannten Rechtecks möglichst groß wird
- der Flächeninhalt des zwischen diesen 4 Punkten aufgespannten Rechtecks möglichst klein wird

Analog Aufg.6c, aber gesucht sind vier Punkte, so dass

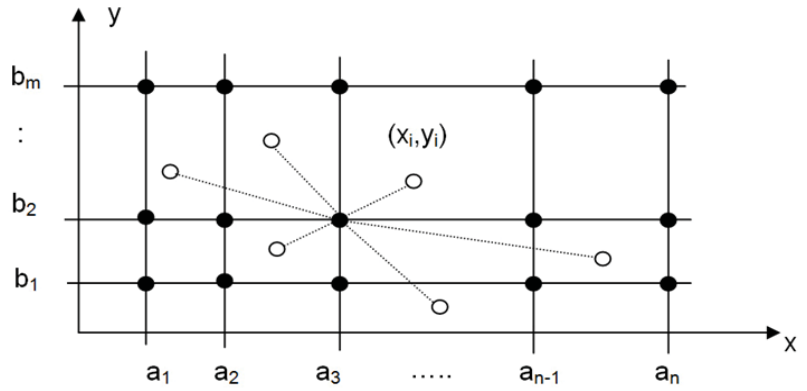
- die 4 Seitenlängen des zwischen diesen 4 Punkten aufgespannten Rechtecks möglichst gleich groß sind, d.h. dass dieses Rechtecks möglichst gleich lange Seiten besitzt
- das zwischen diesen Punkten aufgespannte Rechteck möglichst gleichlange Diagonalen besitzt, d.h. einer quadratischen Form möglichst nahe kommt.

Aufg.7: Abstandssummenproblem in der Ebene

Entwerfen Sie einen **Algorithmus** zu dem folgenden Abstandssummenproblem:

Gegeben seien die Koordinaten (x_i, y_i) mit i = 1(1)k von k Baustellen (bzgl. eines kartesischen Koordinatensystems).
Außerdem seien durch die beiden Vektoren $a = (a_1, a_2, \dots, a_n)^T$,
$b = (b_1, b_2, \dots, b_m)^T$

entsprechend der Skizze die Koordinaten von n*m Gitterpunkten definiert.



Gesucht sind die Koordinaten desjenigen Gitterpunktes (als potenzieller Lagerstandort), für den die Summe der euklidischen Abstände zu allen gegebenen Baustellen (x_i, y_i) mit i = 1(1)k minimal ist.

Hinweis: Nach dem Satz von Pythagoras ergibt sich die Entfernung e eines Gitterpunktes mit den Koordinaten (xx,yy) zu einer Baustelle mit den Koordinaten (x_i, y_i) aus

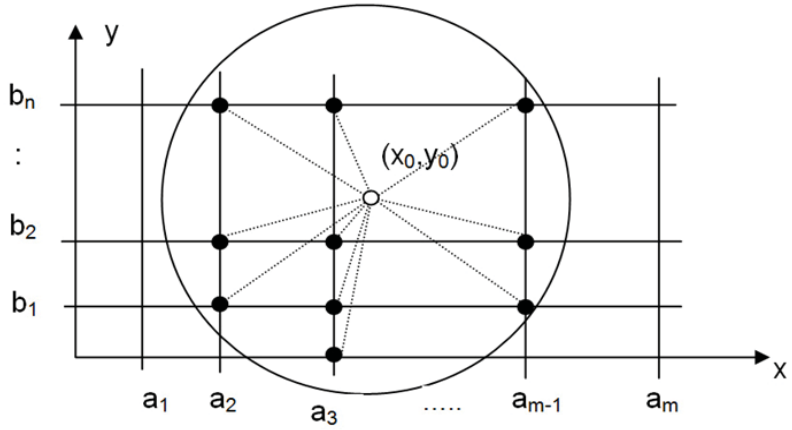
$$e = \sqrt{(xx - x_i)^2 + (yy - y_i)^2}$$

Aufg.8: Abstandssummenproblem in der Ebene

Gegeben seien der Mittelpunkt (x_0, y_0) und der Radius R eines Kreises. Außerdem seien durch die beiden Vektoren

$a = (a_1, a_2, \dots, a_m)^T$, $b = (b_1, b_2, \dots, b_n)^T$

entsprechend der Skizze die Koordinaten von m*n Gitterpunkten definiert.



Gesucht ist die Summe der euklidischen Abstände vom Mittelpunkt des Kreises zu allen innerhalb des Kreises liegenden Gitterpunkten. Bestimmen Sie gleichzeitig, welcher der nicht außerhalb des Kreises liegende Gitterpunkte am nächsten zu dem Rand des Kreises liegt (d.h. zum Mittelpunkt den größten Abstand besitzt und innerhalb des Kreises oder auf dessen Rand liegt).

Hinweis: Ein Gitterpunkt mit den Koordinaten (xx,yy) liegt genau dann innerhalb des Kreises oder auf dessen Rand, wenn gilt:

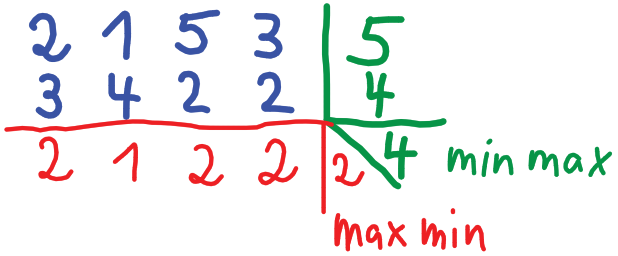
$$(xx - x_0)^2 + (yy - y_0)^2 \le R^2$$

Der Abstand e eines Gitterpunktes (xx, yy) zum Kreismittelpunkt errechnet sich aus:

$$e = \sqrt{(xx - x_0)^2 + (yy - y_0)^2}$$

Aufg.11: MIN MAX = MAX MIN ?

Entwerfen Sie einen Algorithmus in Pseudocode/Struktogramm zur Lösung der folgenden Aufgabe: Gegeben Sei eine mxn-Matrix A.



$$A = (a_{ij})_{i=1(1)m,\,j=1(1)n} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{pmatrix}$$

$\max \min \leq \min \max$

Für jede Zeile i bezeichne $zmax_i$ das Maximum in dieser Zeile und ZM den zugehörigen Vektor dieser Zeilenmaxima.

$$ZM = (z\max_i)_{i=1(1)m}$$

Außerdem bezeichne $smin_j$ das Minimum in der j-ten Spalte und SM den zugehörigen Vektor dieser Spaltenminima.

$$SM = (s\min_j)_{j=1(1)n}$$

Bestimmen Sie zunächst auf algorithmischem Weg die beiden Vektoren *ZM* und *SM*. Falls nun der kleinste Wert des Vektors *ZM* gleich dem größten Wert des Vektors *SM* ist (d.h. das Minimum aller Zeilenmaxima gleich dem Maximum aller Spaltenminima ist), so soll der Algorithmus ein "OK" andernfalls ein "NOOK" ausgeben!

Alg.: IN: m, n, a_{ij} $i = 1(1)m$, $j = 1(1)n$

```
MINMAX = 1E30      FOR (i=1; i<=m; i++) DO      // Zeilen durchlaufen
                     ZMAX_i = -1E30;
                     FOR (j=1; j<=n; j++) DO    // Spalten durchlaufen
                       IF (a_ij >= ZMAX_i) THEN ZMAX_i = a_ij;
IF MAX < MINMAX THEN
MINMAX = MAX
überflüssig {  MINMAX = 1E30;   //alternativ  MINMAX = ZMAX_1
               FOR (i=1; i<=m; i++) DO
                 IF (ZMAX_i < MINMAX) THEN MINMAX = ZMAX_i;
MAXMIN = -1E30
               FOR (j=1; j<=n; j++) DO
                 SMIN_j = 1E30
                 FOR (i=1, i<=m; i++) DO
                   IF (a_ij < SMIN_j) THEN SMIN_j = a_ij
IF (MIN > MAXMIN)
THEN
MAXMIN = MIN
überflüssig {  MAXMIN = -1E30;  //alternativ  MAXMIN = SMIN_1
               FOR (j=1; j<=n; j++) DO
                 IF (SMIN_j > MAXMIN) THEN MAXMIN = SMIN_j;
IF (MINMAX = MAXMIN) THEN OUT: "OK" ELSE OUT: "Nook";
```