Opgave A

Matchen van strings

Invoer file : MATCH.IN
Uitvoer file : MATCH.UIT
Programma : MATCH.PAS of MATCH.C

Gegeven wordt een (niet lege) string met de symbolen uit

{'a','b','c',...'y','z','0','1','2',...,'9'}

met maximaal 30 symbolen. Verder worden er een aantal tekststrings gegeven. Deze strings bestaan uit de symbolen

{'a','b','c',...,'z'}

met maximaal 50 symbolen.

Van deze tekststrings moet bepaald worden of deze met de gegeven tekststring 'matchen'. Hierbij 'matcht' een tekststring met de gegeven string als er voor de cijfers, die in de gegeven string voorkomen, letters ingevuld kunnen worden, zó dat de te matchen string verkregen wordt. Hierbij geldt het voorbehoud dat een cijfer door nul of meer letters vervangen mag worden, waarbij dit aantal **maximaal** de waarde mag zijn welke door het cijfer voorgesteld wordt.

Voorbeelden:

fgalaz matcht *3a1a2z*
fgfgalz matcht niet *3a1a2z*
fgalaz matcht *3a1a23z*
abcdef matcht *3abcdef*
abcdef matcht *3cdef*
abcdef matcht niet *3ef*

De invoer bestaat uit een aantal regels. Op de eerste regel staat het aantal runs vermeld. Iedere run bestaat uit een aantal regels. Op de eerste regel staat de gegeven string, op de tweede regel het aantal te matchen tekststrings, gevolgd door het aantal tekststrings.

De uitvoer bestaat uit de tekst *Run* met het nummer van de run. Hierna volgt voor iedere tekststring de tekst *Matcht* of *Matcht niet* afhankelijk van het feit of ze matchen.

Invoer:
```
<#runs: r>
<gegeven string, run 1>
<#te matchen strings: n1>
<te matchen string 1>
..
<te matchen string n1>
..
..
<gegeven string, run r>
<#te matchen strings: nr>
<te matchen string 1>
..
<te matchen string nr>
<EOF>
```

Uitvoer:
```
run 1
Matcht (niet)
..
Matcht (niet)
..
..
run r
Matcht (niet)
..
Matcht (niet)
<EOF>
```

Voorbeeld invoer:
```
2
3ab3ab3c3
4
abbaabbaabca
bbaabbaabcc
abcababababca
ababcabde
9x9
3
abcdefghi
xyz
x
```

Voorbeeld uitvoer:
```
Run 1
Matcht niet
Matcht
Matcht
Matcht niet
Run 2
Matcht niet
Matcht
Matcht
```

Opgave B

Videobanden

Invoer file : VIDEO.IN
Uitvoer file : VIDEO.UIT
Programma : VIDEO.PAS of VIDEO.C

Een persoon is verslaafd aan speelfilms. Gelukkig bezit hij een videorecorder, waardoor hij deze kan verzamelen. Iedere week huurt hij een aantal films en tapet deze over op videobanden. De films zijn onder te verdelen in de volgende categorieën: Thrillers, Humor, Westerns, Romantiek, Politiefilms en Documentaires. De banden waar de films op komen te staan, zijn van het type 45 min, 1:30 uur, 2 uur en 3:55 uur. De banden kosten respectivelijk *f*4,95 *f*9,95 *f*12,95 *f*24,95. De man streeft naar kostenminimalisatie. Bovendien wil hij op één videoband nooit twee films uit dezelfde categorie hebben staan.

Gevraagd wordt een programma te maken dat voor deze man uitzoekt hoeveel videobanden van welk type hij moet kopen en welke films dan op welke band moeten komen te staan. Voor iedere 'lichting' films koopt hij een aantal (lege) videobanden, dus videobanden uit week 1 waarop nog ruimte vrij is worden niet gebruikt in week 2.

De invoer heeft het volgende formaat. Eerst staat aangegeven over hoeveel weken de optimalisatie uitgezocht dient te worden. Op de volgende regel staat aangegeven hoeveel films er in die week worden gehuurd. Vervolgens staat per regel een film vermeld. Een filmregel bestaat achtereenvolgens uit de eerste letter van de categorie waartoe de film behoort, een spatie en de tijdsduur, waarbij de tijdsduur in het formaat 'u:mm' staat aangegeven (bijv. 1 uur en 12 minuten wordt '1:12', dus 4 karakters). Hierna volgt wederom een spatie en de titel (maximaal 40 karakters lang).

De uitvoer bestaat uit de week met bijbehorende nummer, gevolgd door het aantal banden van het type 45 min, 1:30 uur, 2 uur en 3:55 uur. Er wordt afgesloten met het aantal minuten band dat ongebruikt is en een extra lege regel.

Invoer:
  <aantal weken: w>
  <aantal films, week 1: f_1>
  <1e letter categorie><spatie><u:mm><spatie><titel 1><EoLn>
  ..
  <1e letter categorie><spatie><u:mm><spatie><titel f_1><EoLn>
  ..
  <aantal films, week w: f_w>
  <1e letter categorie><spatie><u:mm><spatie><titel 1><EoLn>
  ..
  <1e letter categorie><spatie><u:mm><spatie><titel f_w><EoLn>
  <EOF>
Uitvoer:
  Week 1
  <m:2> banden van 0:45 uur
  <n:2> banden van 1:30 uur
  <p:2> banden van 2:00 uur
  <q:2> banden van 3:55 uur
  Er zijn in totaal <t:4> minuten tape ongebruikt.
  <lege regel>
  ..
  Week w
  <m:2> banden van 0:45 uur
  <n:2> banden van 1:30 uur
  <p:2> banden van 2:00 uur
  <q:2> banden van 3:55 uur
  <lege regel>
  <EOF>
Voorbeeld invoer:
```
2
3
T 02:13 Friday the 13th, part 20
H 00:10 Tom and Jerry
W 01:33 Zo'n Clint Eastwood filmpje
2
D 02:00 Voortplanting van de Homo Sapiens
P 01:40 Beverly Hills Cop
```

Voorbeeld uitvoer:
```
Week 1
 0 banden van 0:45 uur
 0 banden van 1:30 uur
 1 banden van 2:00 uur
 1 banden van 3:55 uur
Er zijn in totaal  ??? minuten tape ongebruikt.

Week 2
 0 banden van 0:45 uur
 0 banden van 1:30 uur
 2 banden van 2:00 uur
 0 banden van 3:55 uur
Er zijn in totaal   20 minuten tape ongebruikt.
```

Opgave C

Grenspalen

Invoer file : GRENS.IN
Uitvoer file : GRENS.UIT
Programma : GRENS.PAS of GRENS.C

De grenscontroles mogen dan verdwenen zijn, toch willen we nog wel eens weten in welk land een bepaalde plaats ligt. Schrijf een programma dat de grenzen van een aantal landen inleest, en vervolgens van een aantal plaatsen bepaalt in welk land ze liggen. We nemen aan dat alle landen samenhangend zijn, dat wil zeggen er zijn geen eilanden en enclaves. Grenzen worden gespecificeerd door achtereenvolgens gegeven hoekpunten, waarbij het laatste gegeven hoekpunt weer verbonden is met de eerste. Twee hoekpunten worden verbonden door rechte lijnstukken. Zowel plaatsen als hoekpunten worden gegeven door x- en y-coordinaten: gehele getallen tussen -15000 en 15000. Plaatsen liggen nooit precies op een grens. De invoer zit er als volgt uit:

aantal_landen (n>=1;n<=2-)
aantal_hoekpunten_van_land[1] (h_1 >= 3)
<x,y>_van_hoekpunt[1]_van_land[1]
...
<x,y>_van_hoekpunt[h_1]_van_land[1]
aantal_hoekpunten_van_land[2] (h_2 >= 3)
<x,y>_van_<x,y>_van_hoekpunt[1]_van_land[2]
...
<x,y>_van_hoekpunt[h_2]_van_land[2]
...
...
aantal_hoekpunten_van_land[n] (h_n >= 3)
<x,y>_van_hoekpunt[1]_van_land[n]
...
<x,y>_van_hoekpunt[h_2]_van_land[n]
aantal_plaatsen (p >= 1)
coordinaten_van_plaats[1]
...
coordinaten_van_plaats[p]

Verder is gegeven dat het totaal aantal hoekpunten van alle landen samen <= 10000 is. De uitvoer bestaat uit p regels, die het nummer van het land geven waarin de gevraagde plaats ligt, of 0 als de plaats in geen enkel land ligt.

Voorbeeld invoer:

```
3
3
  0   0
100   0
100 100
4
100  0
100 100
200 100
150   0

3
  0   0
100 100
  0 200
3
 15  10
100 110
 50 120
```

Voorbeeld uitvoer:

```
1
0
3
```

Opgave D

Kortste cykel

Invoer file : CYKEL.IN
Uitvoer file : CYKEL.UIT
Programma : CYKEL.PAS of CYKEL.C

Een graaf is een verzameling punten met lijnen ertussen. Een lijn tussen twee punten heet een verbinding. In een ongerichte graaf kun je, als er een pad tussen twee punten bestaat zowel van het ene punt naar het andere lopen, als weer terug. Een cykel is een pad dat rond gaat; d.w.z. je kunt van een beginpunt via tussenliggende punten weer terugkomen bij het beginpunt, zonder twee keer over dezelfde verbinding te komen.

Op de input staat een ongerichte graaf gedefinieerd en wel als volgt:

- een regel met het aantal problemen in het file (=N)
- een regel met het aantal punten van de graaf (=P_1, max. 20)
- P_1 regels, ieder met een genummerd punt gevolgd door zijn buren, gescheiden door spaties. Tussen elk tweetal aangrenzende punten loopt maximaal één verbinding.

Schrijf een programma dat, gegeven deze input, uitvoert wat de lengte van de kortste cykel (kring) in deze graaf is. Als de graaf geen cykels bevat, moet dat overeenkomstig gemeld worden.

Voorbeeld
invoer

```
2
6
1 2 3
2 1 3 4
3 1 2 5
4 2 5
5 3 4 6
6 5
3
2
1 3
3
```

uitvoer

```
Lengte kortste kring is 3
Er is geen cykel in deze graaf
```

Opgave E

Vier op een rij

Invoer file : VIERRIJ.IN
Uitvoer file : VIERRIJ.UIT
Programma : VIERRIJ.PAS of VIERRIJ.C

Bij het spel "Vier op een rij" is het de bedoeling om vier stenen van dezelfde kleur "op een rij" te krijgen. Dit kan een vertikale, een horizontale of een diagonale rij zijn. Het spel wordt gespeeld op een bord van zeven (kolommen) bij zes (rijen). Stenen mogen alleen in de onderste nog niet gevulde rij van een kolom gezet worden. We spreken in deze opgave af dat de kleuren waarmee gespeeld wordt wit en zwart zijn en wit het spel begint.
In de Invoerfile staat een stand van het spel "Vier op een rij" in de vorm van 6 rijen van 7 karakters. De rijen zijn afgesloten door het einde van de regel. Iedere rij bevat de karakters ".", "X" en "O". Het karakter "X" staat voor de witte steen, het karakter "O" staat voor een zwarte steen en het karakter "." staat voor een leeg veld. De eerste regel op de invoer beschrijft de bovenste rij van het bord, de tweede de rij daaronder, ... , en de zevende regel invoer beschrijft de onderste rij van het bord. Er mag vanuit gegaan worden dat de invoer correct is; invoer met stenen in b.v. de 5e rij van een kolom, terwijl de vier rijen daaronder leeg zijn, of invoer waarbij te veel zwarte of witte stenen op het bord staan zal niet voorkomen.
Lees de stand van de invoerfile in en speel alle gedwongen zetten voor zwart en wit. Een gedwongen zet is een zet die gespeeld moet worden, omdat de tegenstander anders met z'n volgende zet kan winnen. Mocht bij een stand op het bord blijken dat een kleur gewonnen staat , stop dan met spelen en druk de stelling af gevolgd door de tekst "In deze stelling staat ... gewonnen.", waarbij op de plaats van de puntjes de tekst "wit" of "zwart" ingevuld is (afhankelijk van welke kleur gewonnen staat). Een kleur k staat gewonnen als de andere kleur het vier stenen op een rij krijgen van k niet verhinderen kan en voor het aantal zetten n, dat nog in totaal door beide kleuren samen gespeeld moet worden voordat k vier stenen op een rij heeft, geldt: $0 <= n <= 2$. Mocht bij een stand blijken dat er geen gedwongen zetten meer zijn, en dat er geen kleur gewonnen staat, druk dan alleen de eindstand af. Het afdrukken van de opmerking dat een bepaalde kleur gewonnen staat, gaat voor het spelen van een gedwongen zet.

Voorbeeld invoer:

```
.......
.......
.......
.......
.OO.XXX
```

Voorbeeld uitvoer:

```
.......
.......
.......
.......
.......
XOOOXXX
```

Opgave F: Ishido

```
INVOER FILE  :  ISHIDO.IN
UITVOER FILE :  ISHIDO.UIT
PROGRAMMA    :  ISHIDO.PAS of ISHIDO.C
```

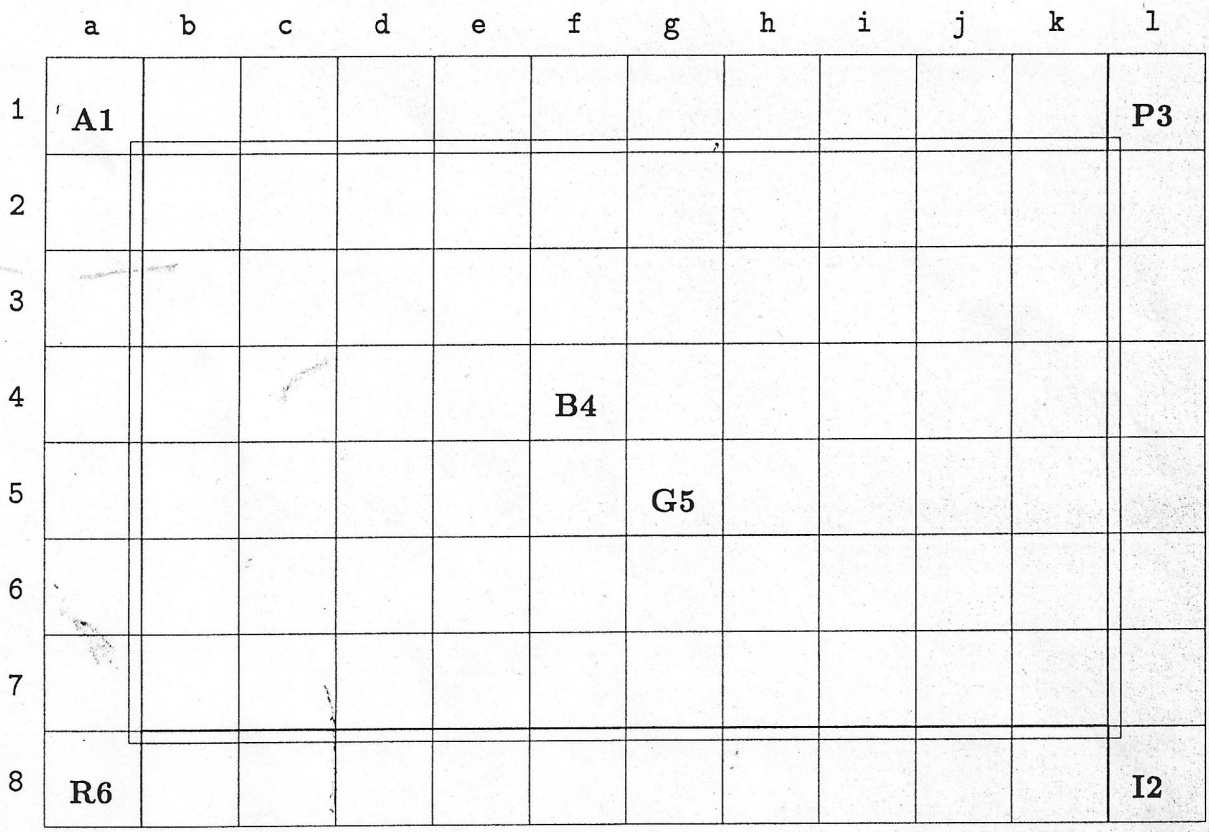
Ishido is een oud Chinees denkspel. Letterlijk vertaald betekent Ishido "de manier van de stenen", en het kan interessante problemen opleveren.

Het spel wordt gespeeld met 72 stenen, twee van elke combinatie van 6 kleuren en 6 symbolen. In deze opgave worden de kleuren Rood, Groen, Blauw, Paars, Indigo en Azuur gebruikt. De symbolen zijn de cijfers 1 tot en met 6. Een steen wordt voorgesteld als de eerste letter van de kleur, en het symbool. R6 is dus een steen van kleur Rood, met 6 als symbool.

Aan het begin van het spel liggen er zes stenen op het bord. Iedere kleur en symbool wordt vertegenwoordigd. Om een steen op het bord te plaatsen, moet die steen bij ten minste één andere (aanliggende) steen "passen", volgens de volgende regels:

- Om bij één steen aan te passen, moet kleur of symbool gelijk zijn.
- Om bij twee stenen aan te passen, moet de kleur met de ene steen gelijk zijn, en het symbool met de andere.
- Om bij drie stenen aan te passen, moeten twee kleuren en één symbool gelijk zijn, òf één kleur en twee symbolen.
- Om bij vier stenen aan te passen, moeten twee kleuren en twee symbolen gelijk zijn.

Dit is een voorbeeld van een beginstand.



Stenen leveren alleen maar punten op als ze niet in de buitenste rand staan.

De score is als volgt:

- Voor aanpassen aan één steen: 1 punt.
- Voor aanpassen aan twee stenen: 2 punten.
- Voor aanpassen aan drie stenen: 4 punten.
- Voor aanpassen aan vier stenen: 8 punten.

Voorbeeld:
Als de volgende steen **A2** is, kan je die op b1 op a2 leggen, maar dat levert je geen punten op. Als de volgende steen **G4** is, kun je die op g4 of f5 kwijt, resulterende in meer punten. Je kunt **G4** natuurlijk ook op h5 of g6 leggen, maar dat levert minder punten op.

De stenen worden in een willekeurige volgorde aangeleverd. De stenen moeten worden gespeeld in de volgorde waarin ze geleverd zijn.

De opdracht is, om een programma te schrijven dat, gegeven een bord en n volgende stenen, bepaalt waar die stenen gespeeld moeten worden om de meeste punten te verdienen.

Voorbeeld:

	a	b	c	d	e	f	g	h	i	j	k	l
1	**A1**											**P3**
2												**P3**
3					**A3**							
4					**B3**	**B4**						
5					**B5**		**G5**	**I5**	**P5**			
6								**I4**				
7		**B6**										**A2**
8	**R6**	**I6**										**I2**

Bovenstaand bord is gegeven. De volgende stenen zijn B5, P4 en R6. De juiste zetten zijn dan:

- Voor **B5**: f5. Aanpassen bij 3 stenen, 4 punten.
- Voor **P4**: i6. Aanpassen bij 2 stenen, 2 punten.
- Voor **R6**: c7, Aanpassen bij 1 steen, 1 punt.

Leg **R6** *niet* aan op a7, want dat is in de buitenste rand, en levert dus geen punten op, hoewel het dan bij twee stenen zou aanpassen, en niet bij één.

In de invoer-file staat een bord, gevolgd door een getal n dat het aantal te leggen stenen aangeeft, en een regel met alle te leggen stenen. Je programma moet dat bord en de stenen inlezen, en bepalen waar die stenen gespeeld moeten worden om de meeste punten te krijgen, en hoeveel punten dat zijn. De uitvoer moet in een tekstbestand komen te staan, met eerst de coördinaten van de plaatsen waar de drie stenen gespeeld moeten worden, gescheiden door spaties, en dan het aantal punten dat dan gescoord wordt.

Voorbeeld:
Invoer:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|A1|  |  |  |  |  |  |  |  |  |  |P3|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |P3|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |A3|  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |B3|B4|  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |B5|  |G5|I5|P5|  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |I4|  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |B6|  |  |  |  |  |  |  |  |  |A2|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|R6|I6|  |  |  |  |  |  |  |  |  |I2|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
3
B5 R4 P6
```

Uitvoer:

```
f5 i6 c7 7
```

Opgave G

Kunstmatige Neurale Netwerken

Invoer file : NEURAL.IN
Uitvoer file : NEURAL.UIT
Programma : NEURAL.PAS of NEURAL.C

Kunstmatige Neurale Netwerken (KNN) zijn netwerken waarin biologische neurons (verregaand gesimplificeerd) worden nagebootst. Deze netwerken worden vooral toegepast bij de ontwikkeling van computers voor beeldverwerking, spraakverwerking, patroonherkenning, enz. De kracht van KNN's ligt vooral in het feit dat KNN's kunnen leren en generaliseren.

In tegenstelling tot gebruikelijke programmeermethodes, waarbij de computer stap voor stap wordt verteld (via een algoritme) wat het moet doen, worden KNN's getraind via een set zorgvuldig geselecteerde leervoorbeelden. Na de training blijkt het netwerk vaak in staat om tevens goede antwoorden te geven voor gevallen die niet letterlijk in de trainingsset voorkwamen!

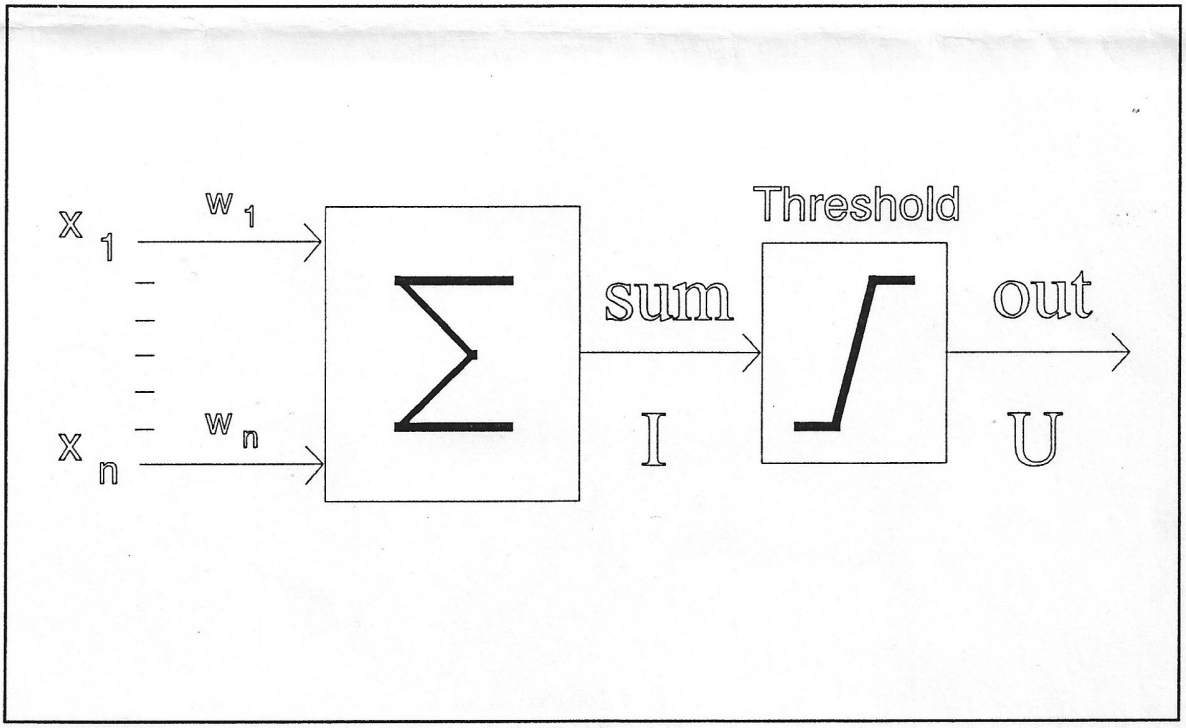


Fig.1: Een enkel neuron

De bouwstenen van Kunstmatige Neurale Netwerken zijn neuronen. Ieder neuron kan een willekeurig aantal invoerkanalen hebben, maar heeft slechts een enkel uitvoerkanaal, dat zich vertakt en de input kan zijn van (zeer) veel andere neuronen. Sommige neuronen in een KNN ontvangen hun invoergegevens rechtstreeks van de buitenwereld (input-neuronen); de 'output-neuronen' bevatten de voor de buitenwereld bestemde resultaten van het netwerk. De functie van het individuele neuron is de volgende: het vormt een gewogen som van zijn inputs, en als deze een zekere drempelwaarde overschrijdt geeft het een output-signaal af. Gedurende de leerfase worden de gewichten, die de relatieve sterkte van de inputs van ieder neuron bepalen, gemodificeerd op basis van een leerregel en de uitwendige gegevens die aan het netwerk worden toegevoerd. In figuur 1 is een neuron schematisch voorgesteld.

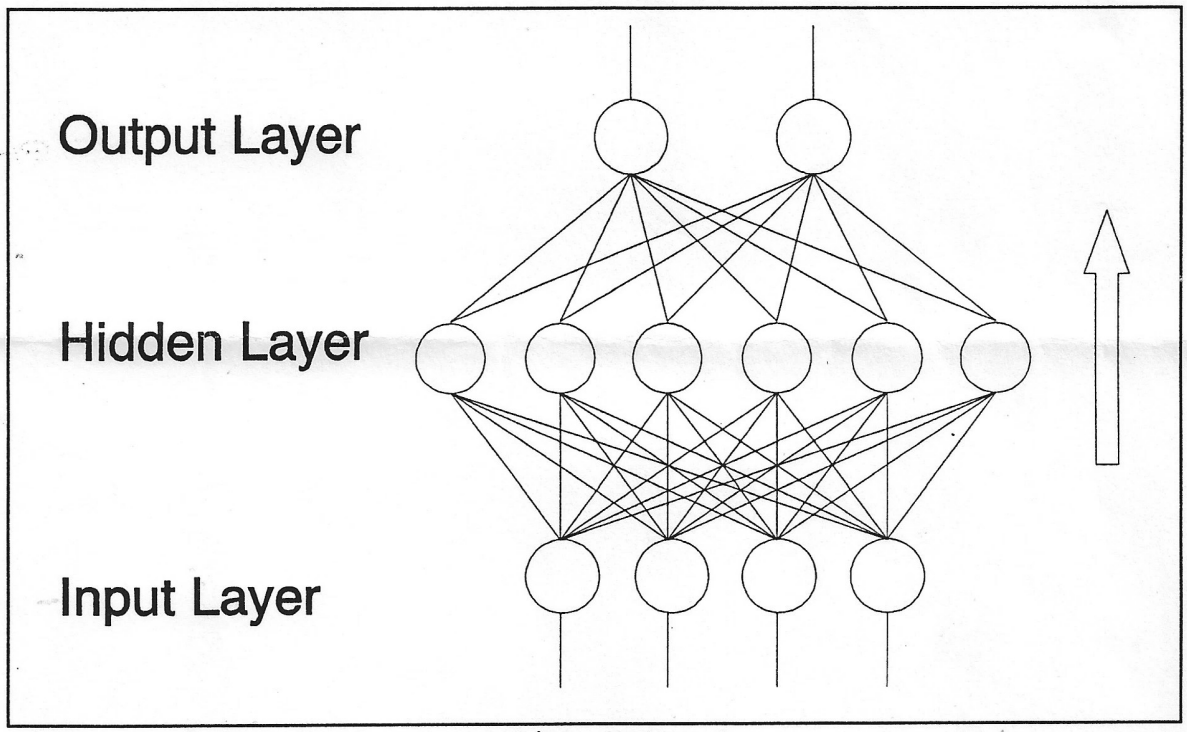


Fig.2: 3-layer Backpropagation Network

Kunstmatige Neurale Netwerken worden tegenwoordig geïmplementeerd met verschillende trainingsalgoritmen. Het meest populaire netwerk is de multilayer backpropagation neurale netwerk (BNN). Deze netwerken hebben een topologie zoals geschetst in figuur 2, alleen hebben ze in het algemeen meer hidden layers. De training is supercised (d.w.z. op basis van de verschillen tussen actuele en gewenste waarde), en tijdens de trainingsfase worden de fouten die optreden in de uitvoerlaag "backpropagated" door het hele net.

In het BNN van figuur 3 hebben de variabelen de volgende betekenis. De variabele $u_i^{\{s-1\}}$ staat voor de momentele output van het i-de neuron in laag s-1. De variabele $w_{ji}^{\{s\}}$ staat voor het gewicht van de connectie tussen het i-de neuron in laag s-1 en het j-de neuron in laag s. De variabele $I_j^{\{s\}}$ staat voor de gewogen som van de inputs naar de j-de neuron in laag s.

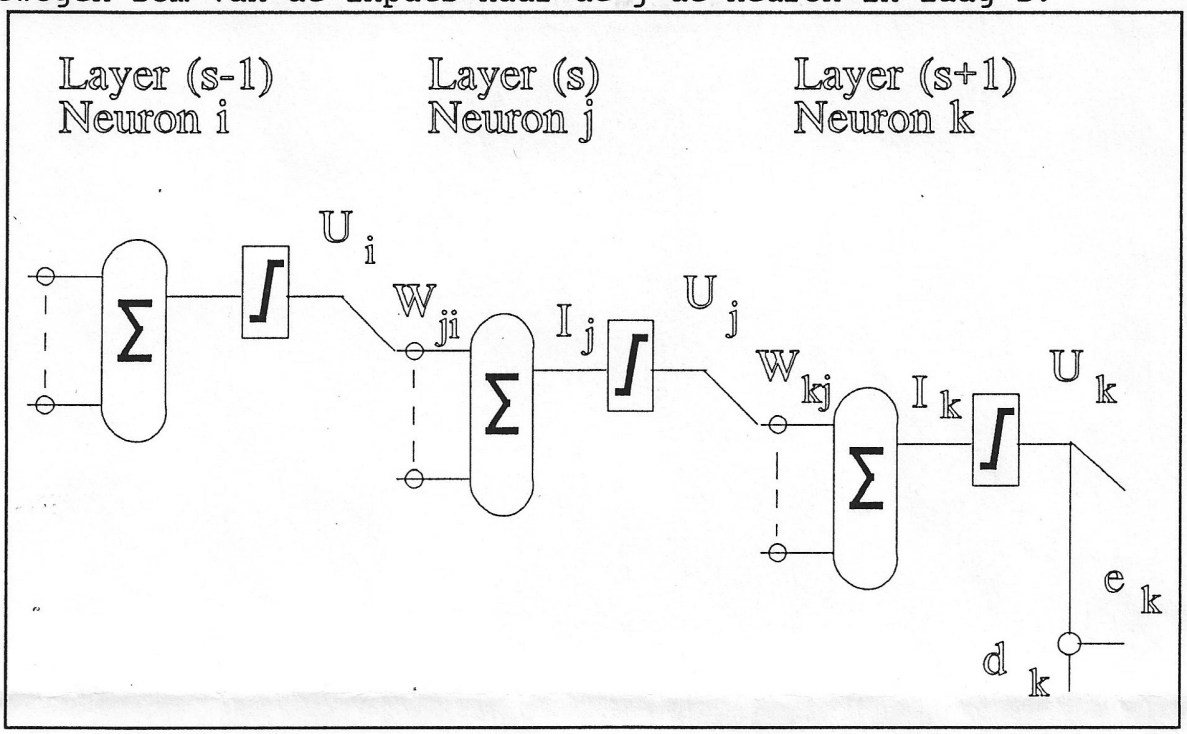


Fig.3 De backprogation network (BNN)

Het proces in een BNN-neuron j in laag s voor het activeren van $u_j^{\{s\}}$ kan als volgt mathematisch worden voorgesteld:

$$U_j^{\{s\}} = g\left(\sum_i U_i^{\{s-1\}} w_{ji}^{\{s\}}\right) = g\left(I_j^{\{s\}}\right) \qquad (1)$$

Hierbij geldt:

$$g(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

welke ook wel bekend staat onder de naam Sigmoid functie.
In de forward-pass worden deze berekeningen uitgevoerd beginnende bij de invoerlaag, via de verborgen lagen naar de uitvoerlaag.

Tijdens de backward pass (in de trainingsfase) worden de gewichten bijgesteld volgens[1]:

$$w_{ji}^{\{s\}}(t+1) = w_{ji}^{\{s\}}(t) + \Delta w_{ji}^{\{s\}}(t+1) \qquad (3)$$

$$\Delta w_{ji}^{\{s\}}(t+1) = \delta_j^{\{s\}} u_i^{\{s-1\}} \qquad (4)$$

Het zogenoemde foutsignaal δ in vergelijking (4) is gedefinieerd voor neuron k in de (output) layer s+1 als

$$\delta_k^{\{s+1\}} = \left(d_k - u_k^{\{s+1\}}\right) \cdot g'\left(I_k^{\{s+1\}}\right) \;;\; d_k - u_k = \epsilon_k \qquad (5)$$

waar

$$g'(x) = \frac{dg}{dx} = g(1-g) \qquad (6)$$

Voor een neuron j in een verborgen laag s is dit foutsignaal gegeven door:

$$\delta_j^{\{s\}} = g'(I_j^{\{s\}}) \sum_k \delta_k^{\{s+1\}} w_{kj}^{\{s+1\}} \qquad (7)$$

Opdracht

Implementeer een backpropagation neuraal netwerk volgens de hierboven beschreven wijze. Vervolgens dient het netwerk getraind te worden met behulp van in het invoerfile gegeven waarden. De invoerfile is als volgt gedefinieerd:

Op de eerste regel worden het aantal tests gegeven (=N), vervolgens worden wat netwerkspecifieke gegevens gegeven met eerst het aantal neuronen in de inputlaag (=i_1), vervolgens het aantal neuronen in een verborgen laag (=h_1), het aantal verborgen lagen (=H_1) en het aantal neuronen in de outputlaag (=o_1). Er zullen geen netwerken voorkomen met meer dan 8 hidden

[1]In deze formule horen ook de learningrate en het momentum thuis. Hiervoor is voor het gemak de waarden 1 respectievelijk 0 gekozen.

layers en in één laag zullen niet meer dan 20 neuronen voorkomen.

Hierna volgt het aantal trainingsregels (=t_1) en t_1 trainingsregels. Een trainingsregel bestaat uit i_1 invoerwaarden gevolgt door o_1 uitvoerwaarden. Hierna volgt het aantal testregels (=T_1) en t_1 regels met i_1 waarden.

In de initiërende fase (dus voordat er ook maar één maal getraind is) dienen de gewichten als volgt gedefinieerd te zijn. De waarde van de gewichten worden per neuron per laag toegekend. Alle gewichten van het eerste neuron in een laag dienen op 0.1 gezet te worden. De gewichten van het tweede neuron in een laag krijgen de waarde 0.2, enz.

De uitvoer bestaat uit T_1 regels met per regel o_1 uitvoerwaarden voor ieder opgegeven Neuraal Netwerk.

Voorbeeld invoer:

Voorbeeld uitvoer: