Gequoteerde zitting Prolog:

NAAM: RICHTING:

Enkele praktische afspraken

- Je krijgt twee uur om deze opdracht **individueel** op te lossen.
- Je raadpleegt **enkel afgedrukte kopies** van de slides (eventueel met handgeschreven nota's) en de ingebouwde manual van SWI-Prolog (gebruik bv. ?-help(write). of ?-apropos(select).)
- In de map **1718_Gequoteerde/Prolog_donderdag** op Toledo vind je de de bestanden `prolog-pipelines.pl`, `prolog-pipelines-facts.pl` en `run.pl`, evenals de indienmodule voor jouw oplossing.
  - Het bestand `prolog-pipelines-facts.pl` bevat de pumpjacks die voorkomen in de voorbeelden in deze opgave. Het bestand `run.pl` can uitgevoerd worden met `swipl -f run.pl` om de voorbeelden uit te voeren. Dit bestand bevat eveneens de verwachte uitvoer voor deze voorbeelden.
  - Als de opdracht expliciet de naam (en ariteit) van een predicaat vermeldt, ben je verplicht om dezelfde naam (en ariteit) te gebruiken in je oplossing.
  - Je oplossing zet je in een bestand `prolog-pipelines.pl` en de eerste lijnen van dit bestand moeten je naam, studentennummer en richting bevatten.

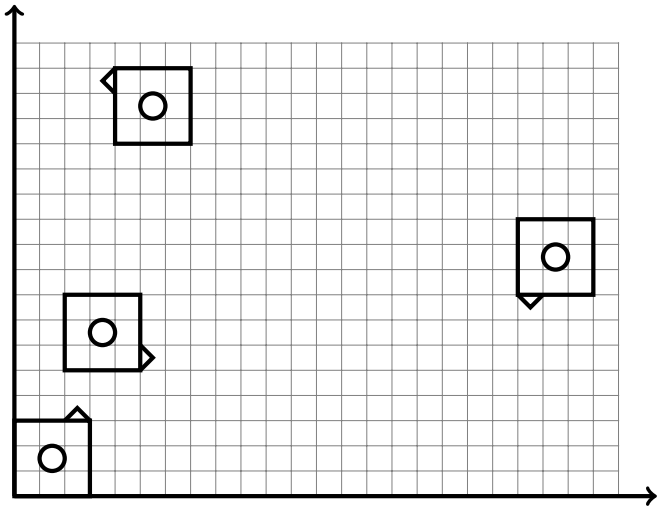
    ```
    % Jan Jansen
    % r0123456
    % master cw
    ```

  - Na twee uur, of wanneer je klaar ben, dien je het `prolog.pl` bestand in via Toledo.

Pipelines

Inleiding

Een oliebedrijf heeft een nieuw olieveld aangeboord. Op dit veld staan reeds een aantal pompen (zgn. "pumpjacks") geïnstalleerd:



Figuur 1: Een aantal pumpjacks in een olieveld.

De pompen hebben een *middelpunt* (aangegeven met een cirkel ○) en een *uitgang* waarop een pijpleiding kan worden aangesloten (aangegeven met een driehoek △). De pompen zijn vanzelfsprekend groter dan hun middelpunt: ze nemen een ruimte van 3x3 vakjes in, zoals het diagram toont.

De positie en oriëntatie van een pumpjacks worden weergegeven door een predicaat `pumpjack/3`. Zo stelt `pumpjack(1,1,north)` de pumpjack links-onder voor, die centrum `(1,1)` heeft, en naar het noorden is georiënteerd.[1] De positieve zin van de x-as (horizontaal) is naar rechts, en de positieve zin van de y-as (verticaal) is naar boven.

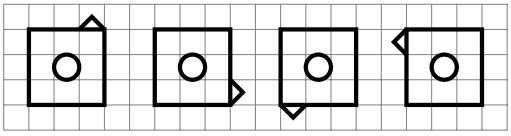
Om de olie op te pompen moeten alle pumpjacks verbonden worden met behulp van pijpleidingen, waarbij zo min mogelijk pijpleiding gebruikt moet worden. De strategie is als volgt: bereken eerst het kortste pad tussen alle

[1] De situatie in Figuur 1 is zoals die in het gegeven bestand.

uitgangen. Plaats vervolgens deze afstanden in een netwerk waarbij iedere node overeenkomt met een uitgang, en iedere edge een gewicht heeft dat overeenkomt met de afstand tussen de twee uitgangen. Bereken tenslotte de *minimal spanning tree* van het netwerk, dit is een netwerk dat alle nodes verbindt, maar alleen die edges overhoudt zodat de som van de gewichten minimaal is.

Deel I: Pumpjacks

De uitgang van een pumpjack kan naar het noorden, het oosten, het zuiden of het westen georiënteerd zijn:



Merk op dat de uitgangen niet gecentreerd zijn, maar bijvoorbeeld in het noordelijke geval 1 vakje naar rechts verschoven is (oost: 1 vakje naar onder, zuid: 1 naar links, west: 1 naar boven).

Opdracht 1 Schrijf een predikaat `pumpjack_output(Pumpjack,OX,OY)` dat als input argument een term van de vorm `pumpjack(X,Y,Direction)` verwacht en slaagt als `OX`- en `OY` de coördinaten van de uitgang zijn.
Bijvoorbeeld:

```
?- pumpjack_output(pumpjack(1,1,north),X,Y).
X = 2, Y = 3.
?- pumpjack_output(pumpjack(1,1,east),X,Y).
X = 3, Y = 0.
?- pumpjack_output(pumpjack(1,1,south),X,Y).
X = 0, Y = -1.
?- pumpjack_output(pumpjack(1,1,west),X,Y).
X = -1, Y = 2.
```

Opdracht 2 Schrijf een predikaat `outputs(Outputs)`, dat geen invoer parameters heeft, en als uitvoer een lijst van alle coördinaten van de outputs van alle pumpjacks die in het bestand `prolog-pipelines-facts.pl` gedefinieerd zijn. De volgorde is niet van belang:

```
?- outputs(Outputs).
Outputs = [(2,3),(5,5),(4,16),(20,7)].
```

Opdracht 3 Schrijf een predikaat `overlaps_pumpjack(Pumpjack,X,Y)` dat waar is als `X` en `Y` coördinaten zijn die vallen binnen de pumpjack voorgesteld door `Pumpjack`. Waarbij `Pumpjack` van de vorm `pumpjack(PX,PY,Direction)` is (`PX` en `PY` zijn de coördinaten van het centrum van de pumpjack). De uitgang is *geen* overlappend deel van de pumpjack, dus `overlaps_pumpjack(pumpjack(1,1,north),2,3)` is bijvoorbeeld `false`.
Bijvoorbeeld:

```
?- overlaps_pumpjack(pumpjack(1,1,north),0,0).
true.
?- overlaps_pumpjack(pumpjack(1,1,east),0,0).
true.
?- overlaps_pumpjack(pumpjack(3,6,east),0,1).
false.
?- overlaps_pumpjack(pumpjack(3,6,west),5,16).
false.
```

Deel II: Buren en Kortste Paden

In dit deel berekenen we het kortste pad tussen de uitgangen. Een pijpleiding kan enkel horizontaal of verticaal verbinden. Nooit *diagonaal*.

Opdracht 4 Schrijf een predikaat `neighbour(X,Y,NX,NY)` dat waar is wanneer `(NX,NY)` onmiddellijk links, rechts, onder of boven `(X,Y)` ligt én niet binnen een pumpjack valt (pijpleidingen kunnen om evidente redenen niet onder een pumpjack liggen).

Bijvoorbeeld:

```
?- neighbour(2,3,NX,NY).
NX = 3,  NY = 3 ;
NX = 1,  NY = 3 ;
NX = 2,  NY = 4 ;
false.
```

Om het kortste pad te vinden gebruiken we het Breadth-First Search algoritme: De te onderzoeken vakjes worden in een First-In-First-Out queue geplaatst. Kijk in iedere stap naar het voorste element van de queue, als dit vakje overeenkomt met de doel-uitgang, dan is het algoritme klaar. Voeg anders de *onbezochte* buren van het vakje achteraan de queue toe, markeer die buren als bezocht (zo komt een vakje nooit meer dan één keer in de queue) en ga naar de volgende stap. Op deze manier bezoekt het algoritme vakjes in oplopende afstand van de start-uitgang, dus wanneer de doel-uitgang gevonden wordt, dan is dit ook het kortste pad.

Opdracht 5 Schrijf een predikaat `shortest_path(Target,Queue,Visited,Distance)` zodat `Target` de coördinaat van de doel-uitgang is, `Queue` een lijst is die een queue voorstelt, `Visited` de lijst van bezochte vakjes is, en `Distance` de lengte van het kortste pad naar `Target` geeft. Het formaat van `Queue` is een lijst van `qi((X,Y),D)` termen, waar `(X,Y)` de coördinaat van het vakje is, en `D` de afstand van dat vakje tot de start-uitgang.
Opgelet: zorg ervoor dat dit predikaat hoogstens één antwoord geeft.
Bijvoorbeeld:

```
?- shortest_path((5,4),[qi((2,3),0)],[],D).
D = 4.
?- shortest_path((1,6),[qi((5,6),0)],[],D).
D = 8.
?- shortest_path((5,4),[qi((2,3),1)],[],D).
D = 5.
```

Een predikaat `shortest_path(Source,Target,D)` met 3 argumenten is reeds voorgedefinieerd.

Opdracht 6 Schrijf een predikaat `shortest_paths(Nodes,Paths)` dat als invoer een lijst `Nodes` van coördinaten van vakjes verwacht, en als uitvoer een lijst van `edge(D,(X1,Y1),(X2,Y2))` termen geeft, waarbij iedere term een pad tussen `(X1,Y1)` en `(X2,Y2)` van lengte `D` voorstelt, voor *iedere* `(X1,Y1)`, `(X2,Y2)` in `Nodes`.
Bijvoorbeeld:

```
?-shortest_paths([(2,3),(5,5)],Paths).
Paths = [edge(5,  (2, 3),  (5, 5)), edge(5,  (5, 5),  (2, 3))].
```

De (zwarte en rode) lijnen in Figuur 2 geven alle verbindingen weer.

Deel III: Minimal Spanning Tree

Herinner u dat een *minimal spanning tree* een netwerk is dat alle nodes verbindt, maar alleen die edges overhoudt zodat de som van de gewichten minimaal is.

Het bepalen van de minimal spanning tree is nu eenvoudig: Alle nodes in het netwerk krijgen aanvankelijk een uniek label. Vervolgens worden alle edges van klein naar groot overlopen. Telkens als een edge twee nodes met verschillende labels verbindt worden de labels gelijkgesteld, en wordt de edge deel van de spanning tree. Als de edge twee nodes met dezelfde label verbindt wordt de edge overgeslagen.

De interface om labels aan te maken, te vergelijken en gelijk te stellen bestaat uit de volgende (voorgedefinieerde) predikaten:

- `create_labels(Nodes,Labels)` maakt unieke labels aan voor alle nodes in de lijst `Nodes`.
- `identical_labels(Labels,Node1,Node2)` slaagt als `Node1` en `Node2` dezelfde label hebben.
- `unify_labels(Labels,Node1,Node2)` stelt de labels van `Node1` en `Node2` gelijk. **Merk op:** dit predikaat heeft geen output parameter, en werkt door middel van unificatie.

Bijvoorbeeld:

```
?- create_labels([(1,2),(2,1)],Labels),
   identical_labels(Labels,(1,2),(2,1)).
false.
?- create_labels([(1,2),(2,1)],Labels),
   unify_labels(Labels,(1,2),(2,1)),
   identical_labels(Labels,(1,2),(2,1)).
true.
```

Opdracht 7 Schrijf een predikaat `iterate_edges(Edges,Labels,MSTIn,MSTOut)`, dat een gegeven partiële[2] minimal spanning tree `MSTIn` verder uitbreidt met edges uit een lijst `Edges`, zodat `MSTOut` een volledige minimal spanning tree van het netwerk is. Ga er vanuit dat de `Edges` van klein naar groot gesorteerd zijn, en hetzelfde formaat hebben als in Opdracht 6.
Bijvoorbeeld:

```
?- create_labels([(2,3),(5,5),(4,16)],Labels),
    Edges = [edge(5,(2,3),(5,5)),edge(12,(4,16),(5,5)),
             edge(17,(2,3),(4,16))],
    iterate_edges(Edges,Labels,[],MST).
MST = [edge(12,(4,16),(5,5)),edge(5,(2,3),(5,5))].
```

De volgorde of richting van de edges in het resultaat is onbelangrijk.

Opdracht 8 Schrijf een predikaat `minimal_spanning_tree(Nodes,Edges,MST)`, die de minimal spanning tree van het netwerk gedefinieerd door `Nodes` en `Edges` (in het formaat van Opdracht 6) berekent: sorteer eerst de edges op hun gewicht, maak de labels aan en roep vervolgens `iterate_edges` op met een lege minimal spanning tree.

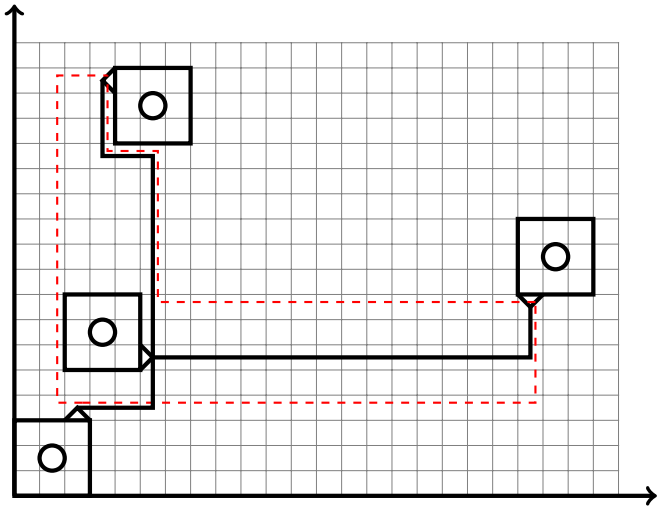
Het (voorgedefinieerde) predikaat `plan(MST)` berekent dan welke pijpleidingen er moeten gelegd worden:

```
?- plan(MST).
MST = [edge(17, (5, 5),  (20, 7)),
```

[2] Dit is minimal spanning tree die nog niet alle nodes verbindt.

```
edge(12, (4, 16), (5, 5)),
edge(5,  (2, 3),  (5, 5))] .
```

De volgorde is niet van belang. Dit plan is afgebeeld in Figuur 2.



Figuur 2: De volle lijn is Het volledige pijpleidingenplan. De gestreepte lijnen zijn de leidingen die geen deel vormen van het plan. **Opgelet:** Dit zijn niet de enige mogelijke paden, maar wel de eenvoudigste—dit wil zeggen met zo weinig mogelijk hoeken. Dit detail is niet van belang aangezien enkel de lengte van het pad relevant is.