

Graded session on Prolog: Steiner trees

NAME:

COURSE:

Some practical points

- This assignment is to be solved **individually** in the next two hours.
- You're only allowed to consult printed copies of the slides (possibly including hand-written notes) and SWI-Prolog built-in manual (using `?- help(write).` or `?- apropos(select).`)
- The map **1718_Graded/Prolog_Wednesday** contains the files `run.pl`, `steinerfacts.pl` and `template.pl`, as well as a solution submission module.
 - File `steinerfacts.pl` contains a small set of facts that is used to generate the example queries in this assignment. File `run.pl` can be run using `swipl -f run.pl`. This runs all the queries specified in this assignment, which enables faster debugging. Finally, `template.pl` can be used as a template for your own solution file.
 - If the assignment explicitly names a predicate name (and arity) for you to implement, use this given name (and arity) in your solution predicate.
 - Your solution should be put into a file `solution.pl` and the first lines should contain your name, student number, and student program.

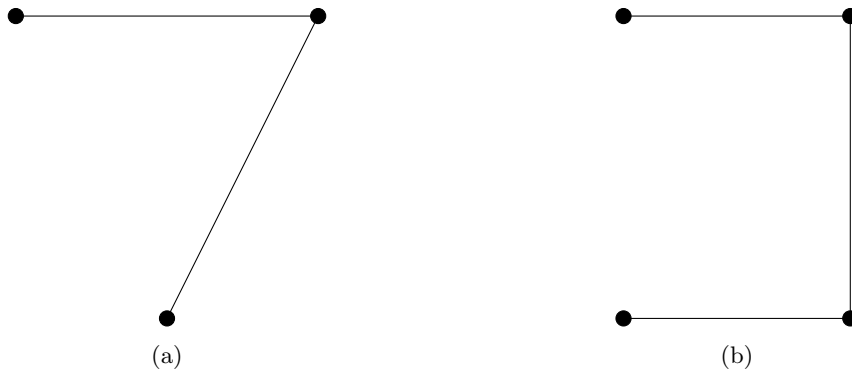
```
% Jan Jansen
% r0123456
% master cw
```
 - After two hours, or when you're finished, submit the `solution.pl` file using Toledo.

Steinerbomen

Inleiding

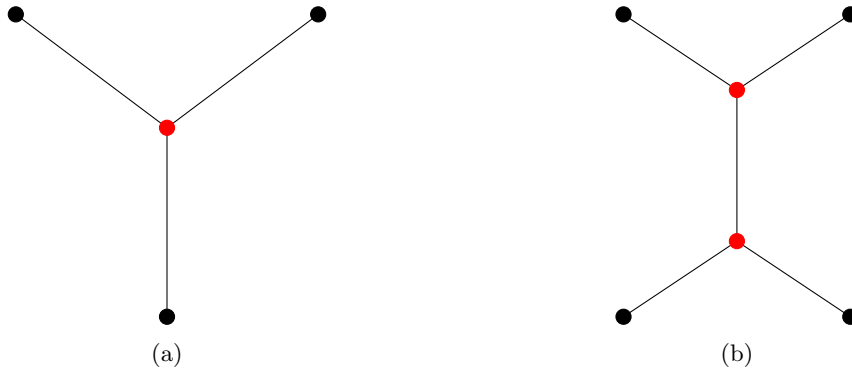
Het minimale Steinerboomprobleem is een wiskundig probleem uit de grafentheorie. Het is een veralgemening van het probleem van de minimaal opspannende boom.

In het probleem van de minimaal opspannende boom zoekt men voor een gegeven verzameling van *knopen* (nodes) een *boom* waarvan de som van de lengten van de *takken* (edges) minimaal is. Een boom is een graaf zonder kringen die alle knopen met elkaar verbindt.



Figuur 1: Minimaal opspannende boom voor (a) 3 en (b) 4 punten.

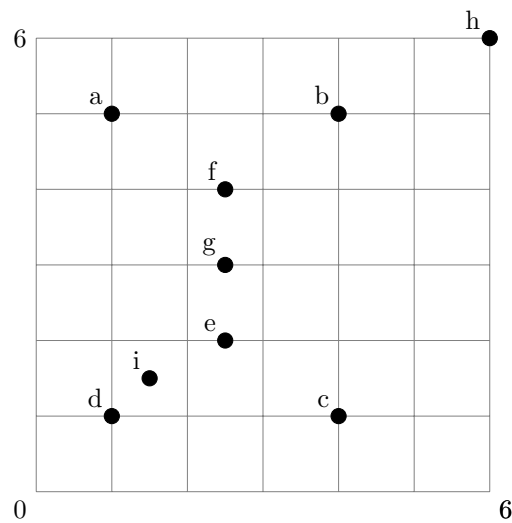
In het Steinerboomprobleem, genoemd naar Jakob Steiner, zoekt men ook voor een gegeven verzameling van punten een boom met de kleinste lengte. Het verschil met het probleem van de minimaal opspannende boom is, dat men nu bijkomende knopen en takken aan de boom mag toevoegen om de lengte ervan nog te verkleinen. Men noemt deze bijkomende knopen *Steinerknopen* en de resulterende graaf is de *Steinerboom*.



Figuur 2: Steinerboom voor (a) 3 en (b) 4 punten. De toegevoegde Steinerpunten zijn in het rood aangeduid.

In deze opgave zullen we zelf een algoritme schrijven om de Steinerboom van een willekeurige verzameling punten in het Euclidisch vlak te vinden. Deze punten worden neergeschreven aan de hand van het `point(S,X,Y)` predicaat.

```
point(a,1,5).
point(b,4,5).
point(c,4,1).
point(d,1,1).
point(e,2.5,2).
point(f,2.5,4).
point(g,2.5,3).
point(h,6,6).
point(i,1.5,1.5).
```



Figuur 3: Visualisatie van de punten in het `steinerfacts.pl` bestand

Oefening 1: Takken

Schrijf een predicaat `edges(Nodes,Edges)`, dat als input een lijst van knopen heeft, en als output een lijst van alle mogelijke takken tussen deze knopen (m.a.w., alle takken van de volledig verbonden graaf). De volgorde is niet van belang. Iedere tak wordt voorgesteld aan de hand van de term `e(A,B,D)` waarbij A en B de knopen voorstellen die zijn verbonden door de tak en D de Euclidische afstand tussen deze twee punten. Let op, deze takken zijn bidirectioneel. Als `e(a,b,3)` al in de outputlijst zit, mag `e(b,a,3)` er niet meer in zitten.

Tip: de euclidische afstand tussen 2 punten (x_1, y_1) en (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

```
?- edges([a,b,c,g],Edges).
Edges = [e(a, b, 3.0), e(a, c, 5.0), e(a, g, 2.5), e(b, c, 4.0), e(b, g, 2.5),
e(c, g, 2.5)].
```

Oefening 2: Sorteren

Schrijf een predicaat `sort_edges(Edges,SortedEdges)` dat edges sorteert op hun lengte. Als verschillende takken dezelfde lengte hebben, zijn er meerdere mogelijkheden waarop de lijst kan gesorteerd worden, maar daar zijn we niet naar op zoek. Dit predicaat mag dus maar 1 lijst als resultaat terug geven. De onderlinge volgorde tussen takken die even lang zijn is niet van belang.

```
?- sort_edges([e(a,c,5.0),e(a,b, 3.0), e(c,g, 2.5)],SortedEdges).
SortedEdges = [e(c, g, 2.5), e(a, b, 3.0), e(a, c, 5.0)].
```

Oefening 3: Pad in bos

Een *bos* is een verzameling van takken waarin er geen kring voorkomt. De takken van een bos kunnen met elkaar verbonden zijn, maar dit is niet noodzakelijk.

Schrijf een predicaat `path(A,B,Forest)` dat slaagt als er een pad bestaat van knoop A naar knoop B dat enkel bestaat uit takken van het gegeven bos (*Forest*).

```
?- path(a,g,[e(a,c,5.0),e(b,c,4.0),e(e,f,2.0),e(b,g,2.5)]).
true .

?- path(b,f,[e(a,c,5.0),e(b,c,4.0),e(e,f,2.0),e(b,g,2.5)]).
false.
```

Oefening 4: Minimaal Opspannende Boom

Schrijf een predicaat `mst(Nodes,Tree)` dat als input een lijst van knopen heeft en als output de minimaal opspannende boom van de gegeven knopen. De minimaal opspannende boom wordt voorgesteld aan de hand van de takken waaruit hij bestaat.

Om de minimaal opspannende boom van een aantal knopen te berekenen, kunnen we het algoritme van Kruskal gebruiken. Dit algoritme gaat als volgt.

1. Creëer een set S die alle mogelijke takken tussen de gegeven knopen bevat.
2. Creëer een bos F dat initieel leeg is.
3. Verwijder de kortste tak t uit S . Als deze tak geen kring vormt met de takken die reeds aanwezig zijn in F , voeg dan t toe aan F . Herhaal deze stap tot er geen kandidaat-takken meer overblijven.
4. Het bos F is nu de minimaal opspannende boom van de gegeven knopen.

```
?- mst([a,b,c,d],Tree).
Tree = [e(a, d, 4.0), e(c, d, 3.0), e(a, b, 3.0)].
```

Oefening 5: Lengte van een boom

Maak een predicaat `treelength(Tree,L)` dat de totale lengte van een boom berekent.

```
?- treelength([e(a, b, 3.0), e(b, c, 4.0)],L).  
L = 7.
```

Intermezzo: Deelverzameling van knopen

In het gegeven bestand `template.pl` vind je het predicaat `subsetk(X,L,K)` dat slaagt als X een deelverzameling is van L met maximaal K elementen. Bijvoorbeeld $[a,c]$ is een deelverzameling van $[a,b,c]$. Elke deelverzameling komt ook maar 1 keer voor. Als bv. $[a,c]$ al werd gegenereerd, komt $[c,a]$ niet meer voor. Dit predicaat kan handig zijn om vraag 6 op te lossen.

```
?- subsetk(X,[a,b,c,d],2).  
X = [a, b] ;  
X = [a, c] ;  
X = [a, d] ;  
X = [a] ;  
X = [b, c] ;  
X = [b, d] ;  
X = [b] ;  
X = [c, d] ;  
X = [c] ;  
X = [d] ;  
X = [] .
```

Oefening 6: Steinerboom

Maak een predicaat `steiner(Nodes,SteinerNodes,K,SteinerTree)` dat voor een verzameling knopen en een verzameling van mogelijke Steinerknopen de minimale Steinerboom terug geeft die maximaal K Steinerknopen bevat. De minimale Steinerboom is de boom met de kleinste totale lengte. Indien 2 Steinerbomen exact dezelfde totale lengte hebben, is de Steinerboom met het kleinste aantal Steinerpunten de meest minimale boom.

```
?- steiner([a,b,c,d],[e,f,g,h,i],1,Tree),treelength(Tree,L).  
Tree = [e(a,i,3.5355339059327378),e(a,b,3.0),e(c,i,2.5495097567963922),  
e(d,i,0.7071067811865476)]  
L = 9.792150443915679.
```

```
?- steiner([a,b,c,d],[e,f,g,h,i],2,Tree),treelength(Tree,L).  
Tree = [e(e,f,2.0),e(d,e,1.8027756377319946),e(c,e,1.8027756377319946),  
e(b,f,1.8027756377319946),e(a,f,1.8027756377319946)]  
L = 9.21110255092798.
```

```
?- steiner([a,b,c,d],[e,f,g,h,i],3,Tree),treelength(Tree,L).  
Tree = [e(e,f,2.0),e(d,e,1.8027756377319946),e(c,e,1.8027756377319946),  
e(b,f,1.8027756377319946),e(a,f,1.8027756377319946)]  
L = 9.21110255092798.
```