CS487 – Introduction to Artificial Intelligence

Winter Semester 2021

Assignment 3
Deadline: 18/01/2022

General instructions

Write a report demonstrating your results using images, screenshots, code and any other explanation necessary. The filename of the deliverable must be in the following form: **ask6_AM** (where AM is your student id number e.g. ask6_1234). Any images/screenshots should be included in your report document. Do NOT submit separate image files. The report document must be in pdf format. Submit a single compressed file (zip/rar/gz etc) containing the report, your Prolog code (**.pl** files) or any other files. Note that this assignment is to be done **individually**. Cheating in any way, including giving your work to someone else, will result in failing this assignment (a mark of zero will be given).

Submit your assignment **electronically** (online) until 18/01/2022 at 23:59 using the course webpage (UoC eLearn platform).

You can use the online environment of SWISH to write and test you Prolog code.

Tasks

Many problems in the field of Artificial Intelligence are reduced to finding a path on a graph. We can represent a directed graph in Prolog (like the one in Figure 1a) using the following facts:

```
connect(a, b).
connect(b, c).
connect(b, d).
connect(e, d).
```
etc …

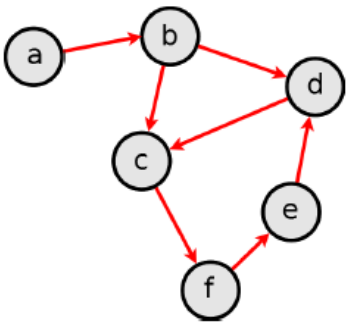

Figure 2a A directed graph

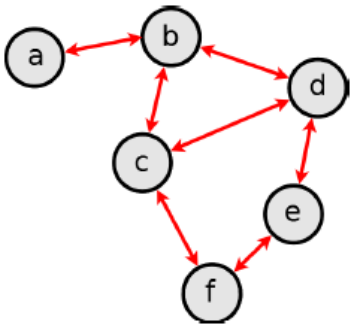

Figure 1b A Non-directed graph

A simple way to represent a non-directed graph (like the one in Figure 1b) would be to double the number of facts to state all the reverse connections (e.g. connect(b, a)). A better way is to define the following two rules:

```
interconnected(X, Y):- connect(X, Y).
interconnected(X, Y):- connect(Y, X).
```

where the predicate `interconnected/2` represents the bi-directional relationship between the nodes.

A. (**15 points**) Could we use just one rule like the following (instead of two rules)?

```
interconnected(X, Y):- interconnected(Y, X)
```

Explain your answer. Are there situations where the above rule would work and other situations that would be problematic?

B. (**25 points**) Define a predicate `exists_path`/2 that implements a simple Depth-First Search. This predicate just confirms the existence of a connection between two nodes. For example, the following question would answer "true":

```
?- exists_path(a, f).
true
```

C. (**35 points**) Define a predicate `path`/3 which will return a list of nodes (i.e. the path) starting from the initial node to the final node. For example, the following question would result in multiple answers:

```
?- path(a, f, Route).
Route=[a, b, c, f];
Route=[a, b, d, e, f];
Route=[a, b, d, c, f];
...
```

D. (**40 points**) Consider the weighted graph shown in Figure 2 where each edge has a specific cost. Modify the facts so that they include the cost information as shown below:

```
connect(a, b, 3).
connect(b, c, 12).
connect(b, d, 4).
 etc …
```

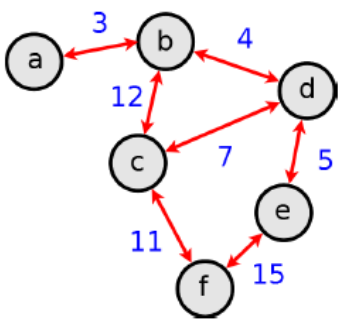

Figure 2 A weighted graph

and the rules:

```
interconnected(X, Y, C):- connect(X, Y, C).
interconnected(X, Y, C):- connect(Y, X, C).
```

Define a predicate `cost_path`/4, which will return a list of nodes starting from the initial node to the final node (like the previous task C) and the total cost of the path. The total cost of the route should be calculated at the same time as finding it. For example, the following question would result in multiple answers:

```
?- cost_path(a, f, Route, Cost).
Route=[a, b, d, c, f],
Cost=25;
Route=[a, b, c, f],
Cost=26;
...
```