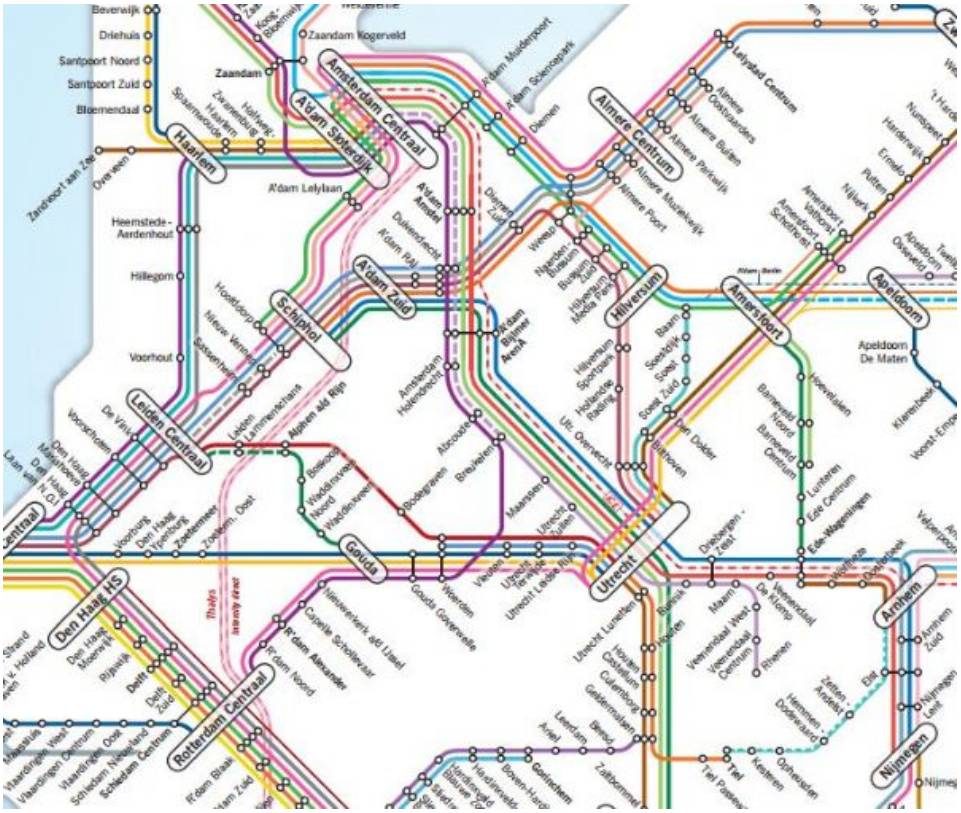
Universiteit van Amsterdam

Programmeertalen

Prolog

Stephan van Schaik - Robin de Vries, Koen van Elsen, Erik Kooistra en Damian Frölich

Reisplanner



Deadline: Zondag 21 februari 2021, 23:59

Mededeling

Belangrijk: voor Prolog zijn de tests op Codegrade niet 'continuous'. Dit houdt in dat je niet meteen feedback krijgt, maar pas na de deadline. Er zijn tests beschikbaar[1] die je lokaal kan runnen en die geven een prima beeld van de tests op Codegrade. Daarbij, als er staat dat je een predikaat moet implementeren en dat predikaat heeft van ons een naam gekregen, houd je dan aan die naam. Als laatste, de volgende predikaten mogen niet worden gebruikt: `halt`, `at_halt`, `initialize` en `initialization`

Tools

Het test framework zit ingebouwd in de standaard `swi-prolog` package. Er wordt gebruik gemaakt van de 'stable' version in de SWI Prolog Ubuntu PPA. Deze zou al op je systeem moeten staan door de BYOD. Zo niet? Kijk dan hier: `https://www.swi-prolog.org/build/PPA.html`

Voor het runnen van de tests is een Makefile aanwezig, kijk daar even in om te zien welk commando je nodig hebt om de tests te runnen.

Puzzles

1. Implementeer `append/3` m.b.v. recursie. Dit predikaat is waar als de derde lijst gelijk is aan de eerste lijst gevolgd door de tweede lijst.
2. Implementeer `palindroom/1` m.b.v. recursie. Dit predikaat is waar als de meegegeven lijst van atomen een palindroom is.
3. Implementeer `sudoku4/2`, een solver voor 4×4 sudoku's. Deze sudoku solver moet bijvoorbeeld de volgende sudoku kunnen oplossen en mag geen externe libaries(zoals clpfd) gebruiken. Het is mogelijk om het hele bord te hardcoden en de regels per cell te beschrijven. Echter, het is een betere oefening om predikaten te schrijven zodat je niet het hele bord hoeft te hardcoden.

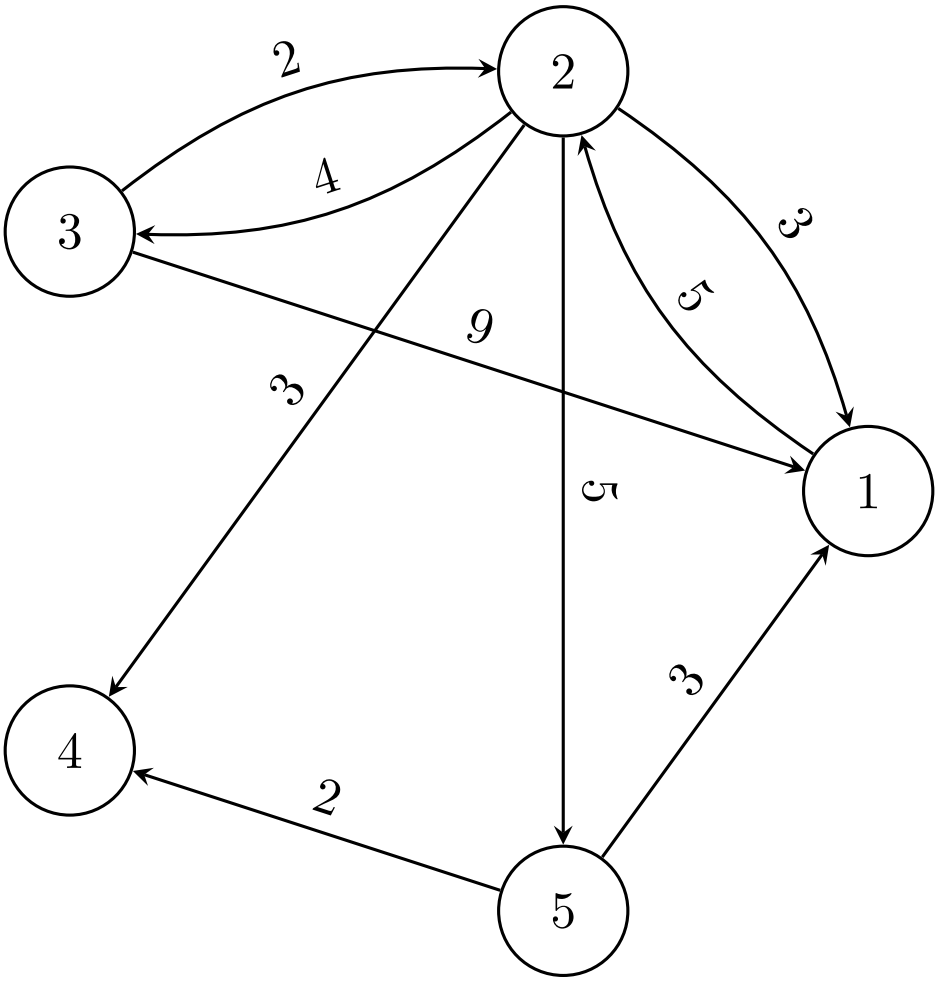
   Shell snippet

   ```
   sudoku4([
       [3,_,4,_],
       [_,1,_,3],
       [2,3,_,_],
       [1,_,_,2]
   ], Solution).
   ```

[1] Op Canvas onder modules -> Prolog -> het tar.gz bestand

Inleiding

Deze week gaan jullie in Prolog een nieuwe reisplanner maken voor de dienstregeling van de Nederlandse Spoorwegen. Voor het overzicht is deze opdracht opgedeeld in verschillende onderdelen. Voor de eerste twee onderdelen hebben jullie de onderstaande graaf nodig:



Deze graaf bestaat uit kanten die de knopen met elkaar verbinden. Iedere kant heeft een bepaald gewicht, de kosten om van een bepaalde knoop naar een naburige knoop te gaan. De volgende fact database representeert deze graaf met regels in de vorm `edge(From, To, Cost)`:

graph.pl

```
edge(1, 2, 5).
edge(2, 1, 3).
edge(2, 3, 4).
edge(2, 4, 3).
edge(2, 5, 5).
edge(3, 1, 9).
edge(3, 2, 2).
edge(5, 1, 3).
edge(5, 4, 2).
```

Opdracht 1: paden zoeken (Ontwikkeld)

Binnen deze graaf moeten er paden gezocht kunnen worden tussen twee willekeurige knopen. Implementeer `path(From, To, Path)` zodanig dat `Path` een pad, een lijst van kanten, aanneemt van `From` naar `To`.

Shell snippet

```
?- path(3, 1, Path).
Path = [edge(3, 1, 9)] ;
Path = [edge(3, 2, 2), edge(2, 1, 3)] ;
Path = [edge(3, 2, 2), edge(2, 5, 5), edge(5, 1, 3)] ;
false.

?-
```

Om het makkelijker te maken wordt het aangeraden om eerst `path(From, To, Visited, Path)` te implementeren, waarbij `Visited` de lijst is van edges die tot dusver bezocht zijn, en `Path` uiteindelijk `Visited` aanneemt, maar in omgekeerde volgorde.

Let bij het zoeken van een naburige knoop in de recursieve stap van `path(From, To, Visited, Path)`, dat deze niet gelijk mag zijn aan `To` en dat deze nog niet bezocht mag zijn.

> Welke operator wordt er gebruikt voor het vergelijken van de naburige knoop met de knoop gegeven door `To` en waarom?

> Waarom mag de betreffende naburige knoop niet onderdeel zijn van een kant die aanwezig is in `Visited`, de lijst van kanten die al bezocht zijn?

> Welke paden zijn er van 1 naar 3, van 3 naar 5 en van 5 naar 4?

Opdracht 2: het kortste pad vinden (Competent)

In het vorige onderdeel hebben we `path(From, To, Path)` geïmplementeerd om de verschillende paden tussen twee knopen te vinden. In dit onderdeel zijn we geïntereseerd in het kortste pad. Hiervoor moeten we eerst kunnen berekenen hoeveel het kost om een bepaald pad te bewandelen. Hiervoor moet `cost(Path, Cost)` geïmplementeerd worden, waarbij `Cost` de kosten zal aannemen om het pad gegeven als `Path` te bewandelen.

Shell snippet

```
?- cost([edge(3, 2, 2), edge(2, 5, 5), edge(5, 1, 3)], Cost).
Cost = 10.

?-
```

> Wat zijn de kosten van ieder pad van 5 naar 4 om deze te bewandelen?

Implementeer nu de regel `shortestPath(From, To, Path)` die het kortste pad geeft tussen twee knopen. Gebruik hiervoor `findall/3` om alle mogelijke paden te vinden en de kosten te berekenen voor ieder pad, en `sort/2` om deze verzamelingen oplossingen te sorteren van goedkoopste naar duurste pad. In sommige gevallen kan het voorkomen dat er meerdere kortste paden zijn met dezelfde kosten maar waarbij het aantal knopen verschillen. Kies in dat geval dan voor het goedkoopste pad met het minste aantal knopen.

Shell snippet

```
shortestPath(3, 1, Path).
Path = [edge(3, 2, 2), edge(2, 1, 3)].
```

> Wat zijn de kortste paden van 1 naar 3, van 3 naar 5 en van 5 naar 4?

Opdracht 3: reistijden integreren (Gevorderd)

In plaats van met grafen te werken, zullen we werken met een aantal trajecten, oftewel paden met de stations en de aankomst- en vertektijd. In Prolog zien deze er als volgt uit:

route.pl:1-14

```
:- op(100, xfx, at).
:- op(50, xfx, :).

route([
    "Amsterdam Centraal" at 11:08,
    "Amsterdam Amstel" at 11:15,
    "Utrecht Centraal" at 11:38,
    "'s-Hertogenbosch" at 12:08,
    "Eindhoven" at 12:32,
    "Weert" at 12:49,
    "Roermond" at 13:02,
    "Sittard" at 13:21,
    "Maastricht" at 13:35
]).
```

Om deze routes te kunnen gebruiken met onze code om het kortste pad te vinden moeten we de regel `edge(From, To, Cost)` zodanig implementeren dat deze twee naburige knopen selecteert uit een pad en de tijdskosten berekent. Om de tijdskosten te kunnen berekenen moet de regel `diffTime(H1:M1, H0:M0, Minutes)` geïmplementeerd worden, waarbij `Minutes` gelijk is aan het aantal minuten dat is verstreken tussen tijd 1 en tijd 0.

Nu kan het pad van Amsterdam Centraal naar Eindhoven opgevraagd worden met de tijd in minuten:

Shell snippet

```
?- shortestPath("Amsterdam Centraal" at Start, "Eindhoven" at End, Path).
Path = [travel("Amsterdam Centraal"at 11:8, "Amsterdam Amstel"at 11:15, 7),
travel("Amsterdam Amstel"at 11:15, "Utrecht Centraal"at 11:38, 23),
travel("Utrecht Centraal"at 11:38, "'s-Hertogenbosch"at 12:8, 30),
travel("'s-Hertogenbosch"at 12:8, "Eindhoven"at 12:32, 24)].

?-
```

> Hoeveel minuten kost het om van Amsterdam Amstel naar Sittard te reizen?

We voegen de volgende route toe aan de database:

route.pl:15-26

```
route([
    "Amsterdam Centraal" at 11:38,
    "Amsterdam Amstel" at 11:45,
    "Utrecht Centraal" at 12:08,
    "'s-Hertogenbosch" at 12:38,
```

```
        "Eindhoven" at 13:02,
        "Weert" at 13:19,
        "Roermond" at 13:32,
        "Sittard" at 13:51,
        "Maastricht" at 14:05
    ]).
```

> Tussen 's-Hertogenbosch en Eindhoven zijn er werkzaamheden. Er worden bussen ingezet tussen deze twee stations. De bus vertrekt vijf minuten na de aankomsttijd, en de rit duurt zo'n dertig minuten. Hoeveel minuten kost het om van Amsterdam Centraal naar Maastricht te gaan?

Opdracht 4: overstappen (Expert)

Om te beginnen voegen we het traject toe van Diemen naar Amsterdam Centraal:

route.pl:28-32

```
    route([
        "Diemen" at 11:06,
        "Amsterdam Science Park" at 11:10,
        "Amsterdam Centraal" at 11:20
    ]).
```

Om te kunnen overstappen op een andere trein moeten we een extra `path/4` predikaat implementeren. Dit predikaat zoekt een edge die vanaf dezelfde knoop begint maar op een later tijdstip. In het pad kan dan naast `travel(From, To, Minutes)` ook `wait(Place, Minutes)` voorkomen:

Shell snippet

```
?- shortestPath("Amsterdam Science Park" at 11:10, "Utrecht Centraal" at _,
Schedule).
Schedule = [travel("Amsterdam Science Park"at 11:10, "Amsterdam Centraal"at
11:20, 10), wait("Amsterdam Centraal", 18), travel("Amsterdam Centraal"at 11:38,
"Amsterdam Amstel"at 11:45, 7), travel("Amsterdam Amstel"at 11:45, "Utrecht
Centraal"at 12:8, 23)].

?-
```

Let op: `wait` kan nooit het eerste of laatste predikaat zijn(zo zijn de tests ingesteld).

Implementeer een `before(Time1, Time2)` predikaat die waar is als Time1 eerder is dan Time2. Gebruik deze in de overstap functie om te zorgen dat er minstens één minuut aan overstaptijd is.

> Wat is het reisplan van Amsterdam Science Park naar 's-Hertogenbosch, hoeveel minuten kost het en hoevaak moeten we overstappen?

Inleveren

De vragen moeten beantwoord worden met zowel queries geschreven in Prolog als formele antwoorden. Deze Prolog queries moeten ingeleverd worden als `queries.pl` en de formele antwoorden moeten ingeleverd worden als `report.pdf`, dat geschreven moet zijn met LaTeX. Als er gebruik is gemaakt van Prolog statements om de vragen te beantwoorden moeten deze in het rapport vermeld staan.

Ook moet er voor iedere opdracht een oplossing ingeleverd worden. De naam hiervan dient in de vorm `solution` gevolgd

door het nummer van de opdracht gevolgd door `.pl` te zijn. Dus, voor de eerste opdracht wordt het `solution1.pl`.

Je kan `make dist` runnen, dit zal een zip file maken met daarin alle benodigde bestanden.

Nogmaals: er mag geen gebruik worden gemaakt van de volgende predikaten: `halt`, `at_halt`, `initialize` en `initialization`