Universidade do Minho

Relatório do Trabalho Prático

SEGMENTO 1

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Sistemas de Representação de Conhecimento e Raciocínio
2º Semestre
Ano Lectivo 2010/2011

54740 Ana Isabel Anjos Sampaio
54746 Miguel Pinto da Costa
54750 Hugo Emanuel da Costa Frade
54772 Tiago Alves Abreu

Braga
11 de Abril de 2011

Resumo

Com a realização do presente segmento do trabalho prático pretende-se a criação um sistema de representação de conhecimento e raciocínio, com capacidade para caracterizar um universo de discurso abordando o tema de localização de serviços. Este tem como objectivo a motivação para a utilização da linguagem de programação em lógica PROLOG na construção de mecanismos de raciocínio para resolver problemas.

Este documento pretende ilustrar, de forma clara e sucinta, todo o trabalho desenvolvido para a resolução dos problemas propostos. É apresentado o estudo que precedeu à sua realização, bem como os resultados obtidos e os respectivas conclusões tiradas após o seu terminado.

ÍNDICE

ÍNDICE DE FIGURAS

1. Introdução

No âmbito da Unidade Curricular (UC) de Sistemas de Representação de Conhecimento e Raciocínio, presente no segundo semestre do curso de Engenharia Informática, de acordo com o plano de estudos do mesmo, foi proposto pelos docentes a realização de um conjunto de trabalhos que abordam o conteúdo leccionado nesta UC. Este relatório é relativo ao segmento 1, de um conjunto de vários segmentos.

Este segmento pretende que seja utilizada a linguagem de programação em lógica PROLOG para a realização de um problema relativo à temática da localização de serviços. Um dos principais objectivos deste segmento é que os alunos se familiarizem com a linguagem PROLOG e com ela consigam resolver problemas de construção de raciocínios.

Neste relatório apresentamos uma proposta de resolução do problema, bem como uma explicação detalhada da mesma. Para isso, dividimos este documento em algumas secções, que consideramos importantes. A primeira secção, Preliminares, destina-se à apresentação do estudo previamente feito para conseguir perceber e realizar este primeiro segmento. O capítulo seguinte é onde descrevemos o teor do trabalho e fazemos a análise de resultados obtidos após a conclusão do mesmo. Por fim, expomos as nossas conclusões e fazemos algumas sugestões de trabalho que podia ter sido feito, no capítulo Conclusões e Sugestões. Encontra-se em anexo o código resultante do trabalho.

2. Preliminares

Neste primeiro segmento do trabalho prático pretende-se abordar a temática da localização de serviços no desenvolvimento de um sistema de representação de conhecimento e raciocínio, com capacidade para caracterizar este universo de discurso.

Neste universo, foram definidos genericamente locais que correspondem a cidades, regiões ou países. Os locais incluem um conjunto de serviços: aeroportos, shoppings, espectáculos desportivos, farmácias, museus, teatros, cinemas e restaurantes.

Existem ligações entre os locais, que inferem sobre o seu tipo (voo, auto-estrada, via municipal, linha ferroviária e caminho rupestre), sobre os locais de origem e destino e, ainda, sobre o custo dessa ligação (distância, em quilómetros, e o valor monetário, em euros).

A proposta de resolução apresentada para o problema é abordada recorrendo a conhecimentos sobre Teoria de Grafos, em que os locais são representados por nodos e as ligações correspondem aos arcos.

3. Descrição do Trabalho e Análise de Resultados

3.1. Base de Conhecimento

Nesta secção é apresentada uma explicação de como estruturamos a nossa base de conhecimento.

3.1.1. Locais

De acordo com o enunciado existem três tipos de locais:

- cidades
- regiões
- países

Como tal, usamos o predicado **local** para descrever estas relações, que possui a seguinte assinatura:

`local(código do local, nome do local, tipo do local).`

Quando analisamos esta assinatura verificamos que o primeiro campo se refere ao código do local, o segundo campo o nome do local e por fim o tipo do local, ou seja, se é cidade, região ou país.

Vejamos um exemplo de cada tipo de local:

- **`local(bra,braga,cidade).`**
  Este exemplo diz-nos que Braga tem o código **`bra`** e é uma cidade.
- **`local(min,minho,regiao).`**
  Este exemplo diz-nos que Minho tem o código **`min`** e é uma região.
- **`local(pt,portugal,pais).`**
  Este exemplo diz-nos que Portugal tem o código **`pt`** e é um país.

Temos também mais um predicado que se chama **regiao** que para uma determinada região indica quais as cidades que lhe pertencem. A assinatura deste predicado é:

regiao(código da região, código da cidade).

ou seja, para cada código da região temos um código de uma das cidades dessa região. Temos por exemplo:

- **regiao(min,bra).**
  Este exemplo diz-nos que a região com código **min** tem as cidades com o código **bra.**
- **regiao(alg,sag).**
  Este exemplo diz-nos que a região com código **alg** tem a cidade com o código **sag.**

Para além destes predicados temos também o predicado `pais` que nos indica, para cada país, quais as regiões que lhe pertencem. Este predicado tem a seguinte assinatura:

pais(código do pais, código da região do país).

Finalmente, analisando a última assinatura usada para definir os locais, vemos que para umcódigo de um dado país temos um código de uma das regiões desse mesmo país. Temos o seguinte exemplo:

- **pais(pt,min).**
  Este exemplo diz-nos que o país com código `pt` tem as regiões com o código `min.`

3.1.2. Serviços

Outro requisito a incluir na base de conhecimento era existir para uma dada cidade um serviço ou um conjunto de serviços. Para representar esta situação temos o predicado `servico` e o predicado `local_servico`. O primeiro tem a seguinte assinatura:

servico(código do serviço, descrição de serviço).

ou seja, para um dado código de serviço diz-nos a sua descrição.

Vejamos o seguinte exemplo:

- **servico(farm,farmacia).**
  Este exemplo diz-nos que o serviço com código **farm** corresponde a **farmacia.**

O segundo tem a seguinte assinatura:

local_servico(código do local, serviço disponível).

Este último é o que interliga o local com os serviços, pois dado o código do local diz-nos um serviço disponível nesse local. Caso pretendamos que esse local tenha mais serviços, temos que acrescentar outra linha equivalente com o serviço pretendido. Um exemplo disto é o que se segue:

- **local_servico(bra,farm).**
  Este exemplo diz-nos que o serviço com código **farm** existe no local com o código **bra.**

3.1.3. Ligações

Para falarmos de ligações primeiramente declaramos o predicado **ligacao** com a seguinte assinatura:

ligacao(código da ligação, descrição da ligação).

Este predicado associa um dado código a uma descrição desse código. Vejamos o seguinte exemplo:

- **ligacao(ae,auto-estrada).**
  Este exemplo diz-nos que o tipo de ligação com código **ae** corresponde a **auto-estrada.**

Existem ainda mais dois predicados sobre as ligações que nos indicam as características de cada ligação entre locais e serviços.

O predicado **ligacaoEntreLocais** tem a seguinte assinatura:

ligacaoEntreLocais(local1,local2,tipo de ligação,custo (distância em km),custo monetário(€))

Esta significa que do **local1** para o **local2** há uma ligação do tipo, com distância e custo consoante o indicado.

Vejamos um exemplo da cidade de braga e ainda para uma melhor ilustração é apresentada na Figura 1 uma representação gráfica dos mesmo:

- **ligacaoEntreLocais(bra,gui,na,30,0).**
  Este exemplo diz-nos que a ligação entre **bra** e **gui** corresponde a **na**, com distância de 30 km e custo de 0€. Os restantes exemplos descrevem-se do mesmo modo.
- **ligacaoEntreLocais(gui,por,ae,40,1).**
- **ligacaoEntreLocais(por,coim,ae,80,2).**
- **ligacaoEntreLocais(por,coim,lf,90,5)**

A representação gráfica deste exemplo está na figura que se segue:

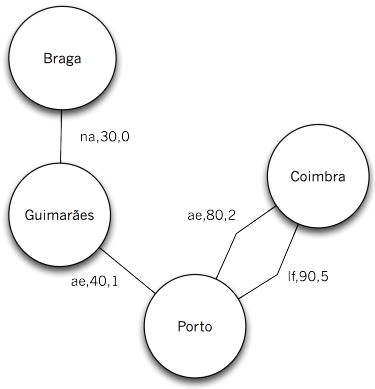

Figura 1 - Grafo representativo do exemplo de ligações entre cidades

O predicado **ligacaoEntreServicos** tem a seguinte assinatura:

ligacaoEntreServicos(serviço1,local1,serviço2,local2, código da ligação, custo de ligação (km), custo monetário(€))

Esta significa que do **serviço1** para o **serviço2** há uma ligação, que inclui a distância e o custo, consoante o indicado.

Vejamos um exemplo da cidade de braga. Para uma melhor ilustração é apresentada na Figura 2 a sua representação gráfica:

- **ligacaoEntreServicos(farm,bra,cine,bra,cr,4,0).**
  Este exemplo diz-nos que a ligação entre **farm** do local **bra** e **cine** do local **bra** corresponde a **cr**, com distância de 4 km e custo de 0€. Os restantes exemplos descrevem-se do mesmo modo.

- **ligacaoEntreServicos(cine,bra,espe,bra,vm,10,0).**
- **ligacaoEntreServicos(cine,bra,shop,bra,vm,12,0).**

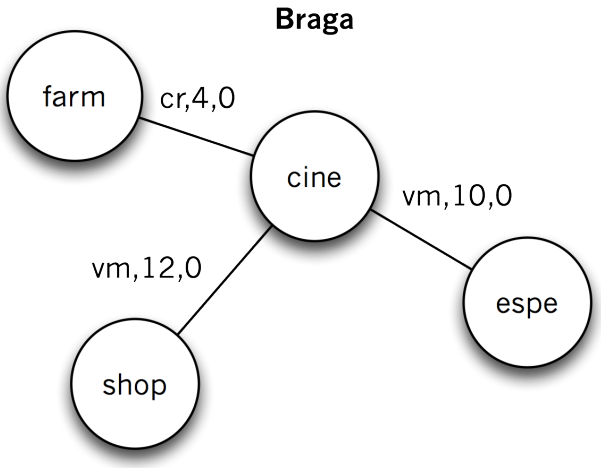

Figura 2 - Grafo representativo do exemplo de ligações entre serviços

3.2. Proposta de resolução para o problema

De acordo com o enunciado, estava prevista a resolução de seis exercícios relativos ao problema, sendo eles:

- Identificar os serviços existentes num determinado local, região ou país;
- Identificar os locais onde esteja presente um determinado serviço ou conjunto de serviços;
- Identificar os serviços que não se podem encontrar numa determinada região;
- Determinar o caminho entre dois locais;
- Determinar o caminho entre dois serviços;
- Determinar o caminho que permite percorrer uma sequência de serviços.

A seguir iremos propor a nossa resoluçãos para os exercícios propostos.

Para além destas propomos também algumas funcionalidades extras.

3.2.1. Exercício 1

De acordo com o enunciado do primeiro exercício, e como já vimos atrás, um local pode ser uma cidade, uma região ou um país. Como tal decidimos incorporar um modo em que o utilizador pudesse ver os serviços existentes ou numa cidade, ou numa região, ou ainda num país. Estes podem ser vistos através dos predicados:

- **servicosCidade**
- **servicosRegiao**
- **servicosPais**

Uma vez que nossa base de conhecimento possui os serviços relativos à cidade em que se encontram, apenas foi necessário procurar todos os serviços relativos a uma determinada cidade e apresenta-los em forma de lista.

Em relação às regiões e países já foi mais difícil. Primeiro tivemos que criar duas novas regras com os seguintes predicados:

- **servicosDeListaDeCidades**
- **servicosDeListaDeRegioes**

A primeira auxiliou-nos em relação ao problema das regiões: para cada cidade da lista de cidades de uma dada região diz-nos quais os serviços que possui. A segunda funcionou do mesmo modo que a primeira mas relativamente aos países: para cada região da lista de regiões de cada país encontra as cidades de cada região e por fim os seus serviços. Contudo, apenas estas duas regras não chegaram e foram necessárias mais algumas para auxiliar. Estas e o respectivo código de toda a base de conhecimento e das regras encontram-se no **Capítulo 6.1 - Código Elaborado**.

3.2.2. Exercício 2

Neste exercício era pedido para dado um serviço ou um conjunto deles identificar-mos todos os locais que possuem esse serviço. Par tal utilizamos a regra que tem o predicado que se segue:

- **determinaLocaisComServicos**

O algoritmo que usamos para a resolução do problema baseia-se em encontrar todos os locais cujo o serviço ou serviços sejam prestados. De seguida estes são colocados numa lista, sendo por fim eliminados os repetidos.

3.2.3. Exercício 3

Neste ponto pretende-se saber quais os serviços que uma dada região não possui. Para resolver este ponto criamos o predicado que se segue:

- **naoExistemServicos**

Este baseia-se em procurar na nossa base de conhecimento todos os serviços existente e coloca-os numa lista. Depois, verifica os que a região possui e coloca-os também numa lista. Por fim, com auxílio da regra **exclusão** conseguimos retirar da lista todos os elementos que pertencem à região, ficando assim apenas com os que não pertencem.

3.2.4. Exercício 4

O objectivo deste exercício é determinar o caminho entre dois locais. Para resolver esta questão utilizamos o seguinte predicado:

- **caminhoEntre2Locais**

que utiliza o predicado auxiliar:

- **travessiaEntre2Locais**

Deste modo, partimos do local inicial, que é colocado imediatamente na lista de locais já visitados. De forma recursiva, o caminho para chegar ao destino vai sendo determinado, e tais locais são directamente incluídos na lista de locais já visitados. O resultado obtido no final (quando é encontrado o destino) corresponde à lista de todos os locais visitados, desde o origem até ao destino.

3.2.5. Exercício 5

Neste exercício pretende-se determinar o caminho entre dois serviços. Para isso criamos os seguintes predicados:

- **caminhoEntre2ServicosMesmaCidade**

Este permite determinar o caminho entre serviços da mesma cidade. O resultado obtido é a lista de serviços dessa cidade por onde se tem que passar para chegar ao serviço destino.

- **caminhoEntre2Servicos**

Este permite determinar o caminho entre serviços de cidades diferentes. O resultado obtido corresponde a uma lista com as cidades por onde foi necessário passar para chegar ao serviço destino.

3.2.6. Exercício 6

O objectivo do último exercício corresponde a determinar o caminho que permite percorrer uma sequência de serviços. Para tal, criamos o predicado:

- **caminhoEntreConjServicos**

que utiliza o predicado como auxiliar:

- **caminhoEntreConjServicosAux**

Estes predicados funcionam em conjunto, da seguinte forma:

1. É determinado o caminho entre os dois primeiros serviços;
2. O seu resultado é colocado numa lista;
3. De forma recursiva, é determinado o caminho para os restantes serviços;
4. No final, quando todos os caminhos entre os serviços já tiverem tratados, é obtida a lista com todos os caminhos resultantes, excluindo os elementos seguidos que se encontrarem repetidos .

3.2.7. Extra 1

Para além da resolução dos exercícios propostos, resolvemos incluir algumas funcionalidades extra ao nosso sistema.

O primeiro extra que incluímos pretende completar o Exercício 4. Trata-se de um predicado que permite o cálculo de um dado caminho, com a designação da distância que separa os dois locais e o custo monetário do percurso.

Segue-se o predicado desta funcionalidade:

- **`caminhoEntre2LocaisCCusto`**

3.2.8. Extra 2

O segundo extra que acrescentamos ao nosso sistema pretende completar o Extra 1, determinando o caminho entre dois locais e, em simultâneo, determina o tipo de ligações (vias) que necessita de tomar, a distância total entre os locais e o preço de tal viagem. Esta funcionalidade permite ao utilizador obter toda a informação relevante na deslocação entre dois locais.

Eis o predicado para esta funcionalidade:

- **`caminhoEntre2LocaisCCustoTipo`**

3.3. Análise de Resultados

Nesta secção apresentamos alguns exemplos de resultados que obtemos quando utilizamos o código que elaboramos.

3.3.1. Exercício 1

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- servicosCidade(por,Servicos).
Servicos = [aero,teat,shop,espe,rest] ?
yes
| ?- servicosCidade(mad,Servicos).
Servicos = [muse,teat,cine,espe] ?
yes

| ?- servicosRegiao(cas,Servicos).
Servicos = [muse,teat,cine,espe] ?
yes

| ?- servicosPais(pt,Servicos).
Servicos = [aero,teat,shop,espe,rest,hosp,farm,cine,shop,espe] ?
yes
```

Como vemos em cima ao executar **servicosCidade(por,Servicos).** estamos a perguntar quais os serviços que nos são prestados pela cidade **por** ao qual obtemos a resposta a **Servicos=[aero,teat,shop,espe,rest]? yes**, ou seja, os serviços listados existem na cidade com código **por**.

De modo análogo fazemos a análise para os restantes casos do exemplo.

3.3.2. Exercício 2

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- determinaLocaisCServicos([aero],Lista).
Lista = [por,lis,bar] ?
yes

| ?- determinaLocaisCServicos([aero,farm],Lista).
Lista = [por,lis,bar,bra,gui] ?
yes
```

Nesta situação temos dois casos. O primeiro quando queremos saber apenas qual os locais onde existe um serviço e o segundo queremos saber os locais onde existem mais que um serviço. É de notar que os locais que dão como resposta no segundo caso são locais que têm ambos os serviços.

3.3.3. Exercício 3

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- naoExistemServicos(min,Serv).
Serv = [restaurante,teatro,aeroporto,museu] ?
yes

| ?- naoExistemServicos(cas,Serv).
Serv = [farmacia,shopping,restaurante,aeroporto,hospital] ?
yes
```

A única interpretação possível deste resultado é que para o primeiro caso na região **min** não existem os seguintes serviços **[restaurante,teatro,aeroporto,museu].** No segundo caso tirámos a mesma conclusão seguindo o raciocínio do primeiro, ou seja, na região **cas** não existem os seguintes serviços **[farmacia,shopping,restaurante,aeroporto,hospital].**

3.3.4. Exercício 4

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- caminhoEntre2Locais(por,gui,Cam).
Cam = [por,gui] ?
yes

| ?- caminhoEntre2Locais(sag,bra,Cam).
Cam = [sag,lis,coim, por,gui,bra] ?
yes
```

Analisando este exemplo vemos que nos é retornada como resposta uma lista com as cidades que são necessárias percorrer para chegar da origem ao destino. Estas respostas mais uma vez vão de encontro com a nossa base de conhecimento. No primeiro caso concluímos que o único caminho possível entre **por** e **gui** é uma única ligação que une estes dois locais. Já no segundo caso vemos que para ir de **sar** a **bra** temos que passar por um conjunto de cidades considerável porque ao contrário do primeiro caso já não existe uma ligação directa entre os dois caminhos.

3.3.5. Exercício 5

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- caminhoEntre2Servicos(aero,por,teat,mad,Cam).
Cam = [por,coim,lis,mad] ?
yes

| ?- caminhoEntre2Servicos(aero,por,espe,por,Cam).
Cam = [espe,aero] ?
yes
```

Neste caso temos uma situação muito particular porque como vemos quando queremos ir de um serviço para o outro mas em locais diferentes, como aparenta o primeiro caso, a resposta será um conjunto de cidades. Caso queiramos encontrar o caminho ente dois serviços dentro do mesmo local vemos que a solução é um conjunto de serviços.

3.3.6. Exercício 6

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?-
caminhoEntreConjServicos([(aero,por),(espe,por),(teat,mad)],Cam).
Cam = [aero,espe,por,coim,lis,mad] ?
yes

| ?- caminhoEntreConjServicos([(aero,por),(teat,mad)],Cam).
Cam = [por,coim,lis,mad] ?
yes

| ?-
caminhoEntreConjServicos([(aero,por),(teat,mad),(espe,por)],Cam).
Cam = [por,coim,lis,mad,lis,coim,por] ?
yes
```

Nesta situação fornecemos um conjunto de serviços para os quais pretendemos saber um caminho. Neste caso, caso algum serviço seja dentro do mesmo local então será indicado a ligação entre serviços. Caso depois continue par outras cidades ou países como no primeiro caso então é juntado à lista o conjunto de cidades que são necessária viajar para chegar ao destino.

3.3.7. Extra 1

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- caminhoEntre2LocaisCCusto(por,mad,Cam,Distancia,Custo).
Cam = [mad,lis,coim,por],
Custo = 126,
Distancia = 2150 ?
yes

| ?- caminhoEntre2LocaisCCusto(por,gui,Cam,Distancia,Custo).
Cam = [gui,por],
Custo = 1,
Distancia = 40 ?
yes
```

Este exemplo é muito simples de entender dado que é uma extensão de um anterior, ou seja, para além de nos dizer o caminho entre dois locais fornece-nos a mais o custo e a distância entre locais. No primeiro caso podemos ver que o custo total da viagem seria de 126€ e que teria de percorrer um total de 2150 km.

3.3.8. Extra 2

Quando executamos as regras definidas no primeiro exercício obtemos o seguinte:

```
| ?- caminhoEntre2LocaisCCustoTipo(por,mad,Cam,TV,Dist,Custo).
TV = [vo,ae],
Cam = [por,coim,lis,mad],
Dist = 2150,
Custo = 126 ?
yes


| ?- caminhoEntre2LocaisCCustoTipo(lis,sag,Cam,TV,Dist,Custo).
TV = [ae],
Cam = [lis,sag],
Dist = 100,
Custo = 5 ?
yes
```

Este extra para além do que o Extra 1 já fornece fornece a mais o(s) tipo(s) de via que são necessários para efectuar o trajecto entre os dois locais. Neste caso no primeiro exemplo temos que os tipos de via utilizados são o tipo **vo** e o tipo **ae**.

4. Conclusões e Sugestões

Após a realização deste primeiro segmento, podemos garantir a interiorizarão dos conhecimentos básicos e fundamentais relativos a esta unidade curricular. Desenvolvemos capacidades no âmbito da representação de conhecimento e raciocino, e também relativamente à utilização da linguagem PROLOG.

No que diz respeito a sugestões para trabalho futuro, seria interessante incluir no sistema funcionalidades capazes que determinar caminhos mais curto entre dois locais distintos, ou o caminho mais económico.

5. Bibliografia

[1] BRATKO, Ivan
"Prolog Programming for Artificial Intelligece"
Addison-Wesley Publishing Company, Jugoslávia, 1986

6. Anexos

6.1. Código elaborado

```
% ------------------- SEGMENTO 1 | Localizacao de servicos -------------------

% --------------------- Base de Conhecimento ---------------------

% Segue-se o conjunto de factos que representam a Base de conhecimento deste
domínio

%Locais
%local(código do local, nome do local, tipo do local (cidade, pais ou região)).

% cidades
% local(cod, nome, cidade)
local(gui,guimaraes,cidade).
local(por,porto,cidade).
local(bra,braga,cidade).
local(lis,lisboa,cidade).
local(coi,coimbra,cidade).
local(bej,beja,cidade).
local(lon,londres,cidade).
local(mad,madrid,cidade).
local(bar,barcelona,cidade).
local(sag,sagres,cidade).

% regioes
% local(cod, nome, regiao)
local(min,minho,regiao).
local(ale,alentejo,regiao).
local(alg,algarve,regiao).
local(cat,catalunya,regiao).
local(cas,castilla,regiao).
local(dou,douro,regiao).

% paises
% local(cod, nome, pais)
local(pt,portugal,pais).
local(es,espanha,pais).
local(fr,fanca,pais).
local(uk,inglaterra).
local(us,eua,pais).
local(br,brasil,pais).

% cidades que pertencem a determinada regiao
% regiao(cod de regiao, cod de cidade)
regiao(min,bra).
regiao(min,gui).
regiao(dou,por).
regiao(alg,sag).
regiao(ale,bej).
regiao(cat,bar).
regiao(cas,mad).
```

```
% regioes que pertencem a determinada regiao
% pais(cod do pais, cod da regiao)
pais(pt,min).
pais(pt,alg).
pais(pt,ale).
pais(pt,dou).
pais(es,cat).
pais(es,cas).

% Servicos disponiveis
% servico(codigo do servico, tipo de servico)
servico(farm,farmacia).
servico(espe,espectaculo_desportivo).
servico(shop,shopping).
servico(rest,restaurante).
servico(cine,cinema).
servico(teat,teatro).
servico(aero,aeroporto).
servico(muse,museu).
servico(hosp,hospital).

% Ligacoes existentes
% ligacao(codigo da ligacao, tipo de ligacao)
ligacao(ae,auto-estrada).
ligacao(lf,linha_ferrea).
ligacao(vm,via_municipal).
ligacao(vo,voo).
ligacao(na,nacional).
ligacao(cr,campo_rupestre).


% Servicos disponiveis num dado local
% servico(local, servico disponivel)
local_servico(bra,farm).
local_servico(bra,cine).
local_servico(bra,shop).
local_servico(bra,espe).
local_servico(gui,espe).
local_servico(gui,farm).
local_servico(gui,hosp).
local_servico(por,aero).
local_servico(por,teat).
local_servico(por,shop).
local_servico(por,espe).
local_servico(por,rest).
local_servico(lis,aero).
local_servico(lis,teat).
local_servico(lis,shop).
local_servico(lis,espe).
local_servico(lis,rest).
local_servico(bar,aero).
local_servico(bar,rest).
local_servico(bar,espe).
local_servico(mad,muse).
local_servico(mad,teat).
local_servico(mad,cine).
local_servico(mad,espe).
```

```
% Ligacoes existentes entre dois locais distintos
% ligacaoEntreLocais(local1,local2,tipo de ligacao,custo (distancia em km),cus-
to monetario (eur))
ligacaoEntreLocais(bra,gui,na,30,0).
ligacaoEntreLocais(gui,por,ae,40,1).
ligacaoEntreLocais(por,coim,ae,80,2).
ligacaoEntreLocais(por,coim,lf,90,5).
ligacaoEntreLocais(coim,lis,ae,70,4).
ligacaoEntreLocais(lis,sag,ae,100,5).

ligacaoEntreLocais(por,lon,vo,600,200).
ligacaoEntreLocais(lis,mad,vo,2000,120).

ligacaoEntreLocais(pt,fr,vo,600,350).
ligacaoEntreLocais(fr,uk,vo,300,500).
ligacaoEntreLocais(uk,us,vo,1500,1000).
ligacaoEntreLocais(us,br,voo,1000,900).

% Ligacoes existentes entre dois servicos distintos
% ligacaoEntreServicos(servico1,local1,servico2,local2, codigo da ligacao, cus-
to de ligacao (km),custo monetario )
ligacaoEntreServicos(farm,bra,cine,bra,cr,4,0).
ligacaoEntreServicos(cine,bra,espe,bra,vm,10,0).
ligacaoEntreServicos(cine,bra,shop,bra,vm,12,0).
ligacaoEntreServicos(farm,gui,espe,gui,vm,7,0).
ligacaoEntreServicos(aero,por,shop,por,ae,5,0).
ligacaoEntreServicos(shop,por,teat,por,vm,13,0).
ligacaoEntreServicos(teat,por,rest,por,vm,2,0).
ligacaoEntreServicos(aero,por,espe,por,ae,10,2).
ligacaoEntreServicos(aero,por,shop,por,ae,5,1).
ligacaoEntreServicos(shop,por,teat,por,vm,13,0).
ligacaoEntreServicos(teat,por,rest,por,vm,2,0).
ligacaoEntreServicos(aero,lis,teat,lis,ae,15,0).
ligacaoEntreServicos(aero,lis,espe,lis,ae,7,1).
ligacaoEntreServicos(espe,lis,shop,lis,cr,1,0).
ligacaoEntreServicos(teat,lis,rest,lis,vm,3,0).
ligacaoEntreServicos(aero,bar,espe,bar,ae,23,3).
ligacaoEntreServicos(aero,bar,rest,bar,vm,5,0).
ligacaoEntreServicos(muse,mad,cine,mad,vm,6,0).
ligacaoEntreServicos(cine,mad,teat,mad,vm,3,0).
ligacaoEntreServicos(cine,mad,espe,mad,ae,25,2).


% ---------------------- Funções Auxiliares ----------------------

% Lista de todas as cidades
listaDeCidades(Cidades):-
      findall(C,local(_,C,cidade),Cidades).

% Lista de todas as regioes
listaDeRegioes(Regioes):-
      findall(R,local(_,R,regiao),Regioes).

% Lista de todos os paises
listaDePaises(Paises):-
      findall(P,local(_,P,pais),Paises).
```

```
% Lista de todos os servicos
listaDeServicos(Servicos):-
	findall(S,servico(_,S),Servicos).

listaDeServicosCodigo(Servicos):-
	findall(S,servico(S,_),Servicos).

% concatenar: L1, L2, L -> {V,F}
concatenar([] , L2 , L2).
concatenar([H1|T1], L2 , [H1|L]) :-
		concatenar(T1,L2,L).

% eliminaRepeticoes: L1,L2 -> {V,F}
eliminaRepeticoes([],[]).
eliminaRepeticoes([H|T],[H|Resto]):-
	negacao(pertence(H,T)),eliminaRepeticoes(T,Resto).
eliminaRepeticoes([H|T],Resto):-
	pertence(H,T),eliminaRepeticoes(T,Resto).

% negacao: Q -> {V,F}
negacao(Q):- Q, !, fail.
negacao(Q).

% exclusao: L1, L2, L -> {V,F}
exclusao([],_,[]).
exclusao([X|L],P,[X|M]):-
	negacao(member(X,P)), exclusao(L,P,M).
exclusao([_|L],P,M):-
	exclusao(L,P,M).

% adicionar: X, L, R -> {V,F}
adicionar(X,[],[X]).
adicionar(X, [H|T] , [X|[H|T]]).

% Verifica se um serviço pertence a uma lista
% pertence: X, L -> {V,F}
pertence(X,[X|L]).
pertence(X,[H|T]) :-
		pertence(X,T).

% procura o nome de um servico dado o codigo
% procuraNomeListaServico: L1, L2 -> {V,F}
procuraNomeListaServico([],[]).
procuraNomeListaServico([X|T],L):-
		servico(X,NomeServ),
		adicionar(NomeServ,[],N),
		procuraNomeListaServico(T,P),
		concatenar(N,P,L).

% procura o nome de um local dado o codigo
% procuraNomeListaLocais: L1, L2 -> {V,F}
procuraNomeListaLocais([],[]).
procuraNomeListaLocais([X|T],L):-
		local(X,NomeLocal,_),
		adicionar(NomeLocal,[],N),
		procuraNomeListaLocais(T,P),
		concatenar(N,P,L).
```

```
% elimina elementos seguidos repetidos de uma lista
% elminaRepSeguidos: Lista1, Lista2, Resposta -> {V,F}
eliminaRepSeguidos([A],L,Resp):-
            concatenar(L,[A],Resp).
eliminaRepSeguidos([H1,H1|T],L,Resp):-
            eliminaRepSeguidos([H1|T],L,Resp).
eliminaRepSeguidos([H1,H2|T],L,Resp):-
            concatenar(L,[H1],R),
            eliminaRepSeguidos([H2|T],R,Resp).


% verifica se existe lugar A->B ou B->A
% ligacaoEntreLocaisComp: LocalA, Local, TipoVia, Distancia, Custo -> {V,F}

ligacaoEntreLocaisComp(X,Y,T,D,C) :-
                  ligacaoEntreLocais(Y,X,T,D,C).

ligacaoEntreLocaisComp(X,Y,T,D,C) :-
                  ligacaoEntreLocais(X,Y,T,D,C).

% verifica se existe servico A->B e B->A
ligacaoEntreServicosComp(A,_,B,_,_,_,_):-
            ligacaoEntreServicos(B,_,A,_,_,_,_).
ligacaoEntreServicosComp(A,_,B,_,_,_,_):-
            ligacaoEntreServicos(A,_,B,_,_,_,_).


% ---------------------- *** Exercicio 1 *** ----------------------

% Identificar os serviços existentes num determinado local, região ou país

servicosCidade(Loc,Servicos):-
            findall(S,local_servico(Loc,S),Servicos).


% Determina os servicos de uma dada regiao
servicosRegiao(Reg,Servicos):-
            findall(C,regiao(Reg,C),Cidades),
            servicosDeListaDeCidades(Cidades,Servicos).


% devolve os servicos de uma lista de cidades
servicosDeListaDeCidades([],L).
servicosDeListaDeCidades([H|T],R) :-
            findall(S,local_servico(H,S),N),
            servicosDeListaDeCidades(T,L),
            concatenar(L,N,P),
            eliminaRepeticoes(P,R).

% Determina os servicos de um dado pais
servicosPais(Pais,Servicos):-
            findall(R,pais(Pais,R),Regioes),
            servicosDeListaDeRegioes(Regioes,Servicos).

servicosDeListaDeRegioes([],L).
servicosDeListaDeRegioes([H|T],R) :-
            servicosRegiao(H,S),
```

```
            servicosDeListaDeRegioes(T,P),
            concatenar(P,S,R).


% --------------------- *** Execercicio 2 *** ----------------------

% Identificar os locais onde esteja presente um determinado serviço ou conjunto
de serviços

% deterninaLocais(Lista de Servicos, Lista de Locais que possuem os servicos)
% determinaLocais2 não se usa
determinaLocais2(X,L) :-
            findall(A,local_servico(A,X),N),
            procuraNomeListaLocais(N,L).

% determina os locais que tem determinados servicos
% determinaLocaisComServicos: L1, L2 -> {V,F}
determinaLocaisComServicos([],[]).
determinaLocaisComServicos([H|T],L) :-
            findall(A,local_servico(A,H),N),
            determinaLocaisComServicos(T,P),
            concatenar(N,P,R1),
            eliminaRepeticoes(R1,L).


% --------------------- *** Execercicio 3 *** ----------------------

% Identificar os serviços que não se podem encontrar numa determinada região

naoExistemServicos: Regiao, Servicos -> {V,F}
naoExistemServicos(Reg,Servicos):-
            listaDeServicosCodigo(ServTotal),
            servicosRegiao(Reg,Serv),
            exclusao(ServTotal,Serv,CodServicos),
            procuraNomeListaServico(CodServicos,Servicos).


% --------------------- *** Execercicio 4 *** ----------------------

% Determinar o caminho entre dois locais

% caminhoEntreLocais: LocalA, LocalB, Caminho -> {V,F}
caminhoEntre2Locais(A, B, Cam) :-
            travessiaEntre2Locais(B, A, [B], Cam).

% travessiaEntre2Locais: LocalA, LocalB, ListaVisitados, NovaListaVisitados ->
{V,F}
travessiaEntre2Locais(A, B, Visitados,[B|Visitados]) :-
            ligacaoEntreLocaisComp(A, B,_ ,_,_).

travessiaEntre2Locais(A, B, Visitados, Cam) :-
            ligacaoEntreLocaisComp(A, C, _, _,_),
            C \== B,
            \+ member(C, Visitados),
            travessiaEntre2Locais(C, B, [C|Visitados], Cam).
```

```
% -------------------- *** Execercicio 5 *** --------------------

% Determinar o caminho entre dois serviços

% caminhoEntre2ServicosMesmaCidade: ServicoA, ServicoB, Caminho -> {V,F}
caminhoEntre2ServicosMesmaCidade(A, B, Cam):-
            travessiaEntre2Servicos(A, B, [A], Cam).
% travessiaEntre2Servicos: ServicoA, ServicoB, ServicosVisitados, NovaListaVi-
sitados -> {V,F}
travessiaEntre2Servicos(A, B, Visitados, [B|Visitados]):-
            ligacaoEntreServicosComp(A, _, B, _, _,_,_).
travessiaEntre2Servicos(A, B, Visitados, Cam):-
            ligacaoEntreServicosComp(A, _, C, _,_,_,_),
            C \== B,
            \+ member(C, Visitados),
            travessiaEntre2Servicos(C, B, [C|Visitados], Cam).

% caminhoEntre2Servicos: Servico1, Cidade1, Servico2, Cidade2, Caminho -> {V,F}
caminhoEntre2Servicos(S1, C1, S2, C1, Cam):-
            local_servico(C1, S1),
            local_servico(C1, S2),
            caminhoEntre2ServicosMesmaCidade(S1, S2, Cam).
caminhoEntre2Servicos(S1, C1, S2, C2, Cam):-
            local_servico(C1, S1),
            local_servico(C2, S2),
            caminhoEntre2Locais(C2, C1, Cam).


% -------------------- *** Execercicio 6 *** --------------------

% Determinar o caminho que permite percorrer uma sequência de serviços

% caminhoEntreConjServicos: ListaDeServicos, Caminho -> {V,F}
caminhoEntreConjServicos(L, Resp):-
            caminhoEntreConjServicosAux(L, Cam, R1),
            eliminaRepSeguidos(R1, [], Resp).

% caminhoEntreConjServicosAux: ListaServicos, Caminho -> {V,F}
caminhoEntreConjServicosAux([(S1,C1)],Cam).

caminhoEntreConjServicosAux([(S1,C1),(S2,C2)],Cam, R1):-
            caminhoEntre2Servicos(S2,C2,S1,C1,Cam1),
            concatenar(Cam, Cam1, R1).

caminhoEntreConjServicosAux([(S1,C1),(S2,C2)|T],Cam, R1):-
            caminhoEntre2Servicos(S2,C2,S1,C1,Cam1),
            concatenar(Cam, Cam1, Resp),
            caminhoEntreConjServicosAux([(S2,C2)|T],Resp, R1).


% -------------------- *** Extras *** ----------------------

% Calculo do custo de uma ligacao

% calculaCusto: tipo de custo, [ligacaoEntreLocais], Custo -> {V,F}
calcularCusto(monetario,[ligacaoEntreLocais(_,_,_,_,C)],C).
calcularCusto(monetario,[ligacaoEntreLocais(_,_,_,_,C)|T],Custo):-
```

```
            calcularCusto(monetario,T,Valor),
            Custo is Valor+C.

calcularCusto(distancia,[ligacaoEntreLocais(_,_,_,D,_)],D).
calcularCusto(distancia,[ligacaoEntreLocais(_,_,_,D,_)|T],Distancia):-
            calcularCusto(distancia,T,Valor),
            Distancia is Valor+D.



% Calcula Caminho entre Locais com o custo, distancia.
% caminhoEntre2LocaisCCusto: LocalA, LocalB, Caminho, Distancia, Custo -> {V,F}
caminhoEntre2LocaisCCusto(A, B, Cam, Distancia, Custo) :-
            travessiaEntre2LocaisV2(A, B, [A], Cam, Distancia, Custo).

% travessiaEntre2LocaisV2: LocalA, LocalB, LocaisVisitados, NovaListaVisitados,
Distancia, Custo -> {V,F}
travessiaEntre2LocaisV2(A, B, Visitados,[B|Visitados], Distancia1, Custo1) :-
            ligacaoEntreLocaisComp(A, B,_,Distancia1,Custo1).

travessiaEntre2LocaisV2(A, B, Visitados, Cam, Distancia, Custo) :-
            ligacaoEntreLocaisComp(A, C,_,Distancia2,Custo2),
            C \== B,
            \+ member(C, Visitados),
            travessiaEntre2LocaisV2(C, B, [C|Visitados], Cam, DistanciaResto,
CustoResto),
            Custo is Custo2 + CustoResto,
            Distancia is Distancia2 + DistanciaResto.


% Calcula Caminho entre Locais com o custo, distancia e o tipo de via associa-
do.
% caminhoEntre2LocaisCCustoTipo: LocalA, LocalB, TipoVia, Distancia, Custo ->
{V,F}
caminhoEntre2LocaisCCustoTipo(A, B, Cam, TipoVia, Distancia, Custo) :-
            travessiaEntre2LocaisV2Tipo(B, A, [B], Cam, R, Distancia, Custo),
            eliminaRepeticoes(R,TipoVia).

% travessiaEntre2LocaisV2Tipo: LocalA, LocalB, LocaisVisitados, NovosVisitados,
TipoVia, Distancia, Custo -> {V,F}
travessiaEntre2LocaisV2Tipo(A, B, Visitados,[B|Visitados], TipoVia, Distancia1,
Custo1) :-
            ligacaoEntreLocaisComp(A, B,TipoVia1,Distancia1,Custo1),
            adicionar(TipoVia1,[],TipoVia).

travessiaEntre2LocaisV2Tipo(A, B, Visitados, Cam, TipoVia, Distancia, Custo) :-
            ligacaoEntreLocaisComp(A, C,TipoVia1,Distancia2,Custo2),
            C \== B,
            \+ member(C, Visitados),
            travessiaEntre2LocaisV2Tipo(C, B, [C|Visitados], Cam, TipoVia2,
DistanciaResto, CustoResto),
            Custo is Custo2 + CustoResto,
            Distancia is Distancia2 + DistanciaResto,
            adicionar(TipoVia1,TipoVia2,TipoVia).
```