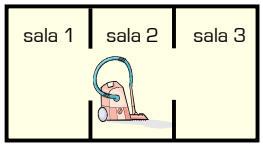
1. Um **fazendeiro** precisa atravessar um rio levando um **lobo**, uma **ovelha** e um **repolho**. Para isto, ele dispõe de um pequeno bote com capacidade para levar apenas ele mesmo e mais uma de suas cargas. Ele poderia cruzar o rio quantas vezes fossem necessárias para transportar seus pertences; porém, na sua ausência, a ovelha pode comer o repolho e o lobo pode comer a ovelha. Complete a descrição de ações a seguir (`volta`, `trazL`, `trazO` e `trazR`) e use busca *aleatória* para obter um plano que possibilite o fazendeiro atravessar o rio sem perder nenhum de seus pertences.

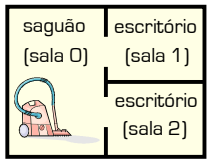
```
% estado indica margem em que cada elemento está
inicial([e,e,e,e]).
meta([d,d,d,d]).

ação(vai,  [e,L,O,R],[d,L,O,R],1) :- L\=O, O\=R.
ação(levaL,[e,e,O,R],[d,d,O,R],1) :- O\=R.
ação(levaO,[e,L,e,R],[d,L,d,R],1).
ação(levaR,[e,L,O,e],[d,L,O,d],1) :- L\=O.
...
```

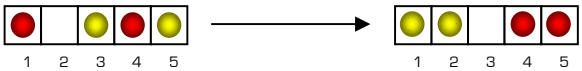
2. Considere uma versão do *Mundo do Aspirador*, como ilustrado ao lado. Estando na sala 1, o agente só pode entrar na sala 2; estando na sala 2, ele pode entrar nas salas 1 ou 3 e, estando na sala 3, ele só pode entrar na sala 2. Um estado do mundo é representado por `[A,S1,S2,S3]`. Por exemplo, o estado em que o agente está na sala 2 e apenas as salas 1 e 3 estão sujas é representado por `[2,s,l,s]`. Com base nesta representação, defina as ações do agente (`entrar1`, `entrar2`, `entrar3` e `aspirar`) e use busca *aleatória*, em *largura* e *profundidade* para encontrar um plano que leve do estado inicial `[2,s,l,s]` ao estado meta `[1,l,l,l]`.



3. Considere outra versão do *Mundo do Aspirador* cujo prédio tem dois pisos, cada um deles contendo um saguão (sala 0) e dois escritórios (salas 1 e 2). Para mudar de piso ou entrar num escritório o aspirador deve estar no saguão. Um estado do mundo é representado por `[SA, PA, S11, S21, S12, S22]`. Por exemplo, o estado em que o agente está no saguão do primeiro piso e apenas as salas 21 e 12 estão sujas é representado por `[0,1,l,s,s,l]`. Usando esta representação, defina as ações do agente (`subir`, `descer`, `aspirar`, `sair`, `entrar1` e `entrar2`) e use busca em *largura* e em *profundidade* para obter um plano que transforme o estado inicial `[0,1,l,s,s,l]` no estado meta `[0,1,l,l,l,l]`.



4. O *Problema das Fichas* consiste num quebra-cabeça composto por um tabuleiro com cinco posições e apenas quatro fichas, sendo duas vermelhas e duas amarelas.

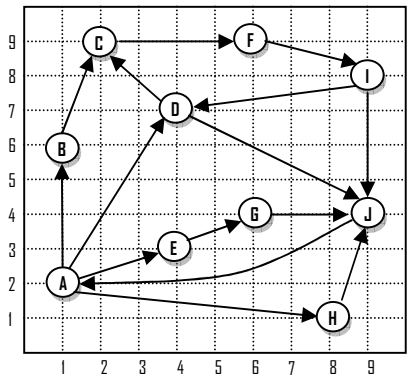


O objetivo é mover as peças de modo que as duas amarelas fiquem à esquerda e as duas vermelhas fiquem à direita. As ações possíveis são da forma `mij`, sendo `i` a posição em que a ficha se encontra e `j` a nova posição para a qual ela é movida (que deve estar livre). Por exemplo, a ação `m21` move a ficha da posição 2 para a posição 1; enquanto a ação `m52` move da posição 5 para a posição 2. Se `i` e `j` são posições vizinhas, o custo da ação é 1; caso contrário, o custo da ação é igual ao dobro do número de posições saltadas pela ficha movida. Por exemplo, o custo da ação `m21` é 1; enquanto o custo da ação `m52` é 4.

Complete a descrição de ações a seguir e use busca *aleatória*, em *largura*, *profundidade* e de *menor custo* para obter um plano que transforme o estado inicial `[v,-,a,v,a]` no estado meta `[a,a,-,v,v]`.

```
ação(m21,[-,A,B,C,D],[A,-,B,C,D],1).
ação(m31,[-,A,B,C,D],[B,A,-,C,D],2).
ação(m41,[-,A,B,C,D],[C,A,B,-,D],4).
ação(m51,[-,A,B,C,D],[D,A,B,C,-],6).
...
```

5. O *Problema das Rotas* consiste em, dado um mapa de vôos entre aeroportos, encontrar uma rota que leve de um aeroporto até outro. O programa a seguir modela este problema, considerando o mapa na figura abaixo.




```
% estado inicial e meta do agente
inicial( a ).
meta( j ).
% ações do agente
ação(voa(P,Q),P,Q,D) :- vôo(P,Q), dist(P,Q,D).
% vôos existentes entre os aeroportos
vôo(a,b). vôo(a,d). vôo(a,e). vôo(a,h).
vôo(b,a). vôo(b,c). vôo(c,f). vôo(d,c).
vôo(d,j). vôo(e,g). vôo(f,i). vôo(g,j).
vôo(h,j). vôo(i,d). vôo(i,j). vôo(j,a).
% posição dos aeroportos no mapa
pos(a,1,2). pos(b,1,6). pos(c,2,9). pos(d,4,7).
pos(e,4,3). pos(f,6,9). pos(g,6,4). pos(h,8,2).
pos(i,9,8). pos(j,9,4).
% cálculo da distância entre dois aeroportos
dist(P,Q,D) :- pos(P,Xp,Yp), pos(Q,Xq,Yq),
               D is sqrt((Xp-Xq)^2 + (Yp-Yq)^2).
% função heurística: distância em linha reta
h(S,H) :- meta(M), dist(S,M,H).
```

- Digite o programa e veja que solução é encontrada por cada um dos tipos de busca (1 a 6).
- O predicado `time(C)` informa o número de inferências realizadas para responder a uma consulta `C`. Use este predicado para comparar a eficiência dos vários tipos de busca (e.g., `?- time(busca(1)).`). O que você conclui:
  - (a) A busca em *profundidade* pode ser mais rápida que em *largura*, mesmo quando ela obtém um plano com mais passos?
  - (b) Um plano com um número mínimo de passos necessariamente tem custo mínimo?
  - (c) Podemos dizer que, em geral, a busca pela *melhor estimativa* é a mais rápida de todas?
  - (d) Qual é a busca mais eficiente, que pode garantir encontrar uma solução de custo mínimo?