1.
a)
Analise os predicados escritos em Mercury apresentados em seguida e explique o que fazem:

```
:- pred primeiro(T, list(T)).
:- mode primeiro(in, in) is semidet.
primeiro(X,[X|_]).
primeiro(X, [_|T])
:- primeiro(X,T).
```

É um predicado semideterministico, que testa se X pertence à lista, caso pertença sucede, caso não pertença falha.
Poderá ser relevante dizer que para quando encontra a 1º ocurrencia.

b)

```
:- pred segundo(list(T)::in,list(T)::out) is multi.
segundo([],[]).
segundo(L1,L2)
:- if delete(L1,X,R1) then segundo(R1,R2), L2=[X|R2] else L2=[].
```

É um predicado multi, sucede sempre e pode ter uma ou mais soluções.
Recebe uma lista e devolve todas as permutações possiveis dos elementos da lista.

c)

```
:- pred terceiro(pred(T), list(T)).
:- mode terceiro(pred(in) is semidet, in) is semidet.
terceiro(_, []).
terceiro(P, [H|T]):- P(H), terceiro(P, T).
```

se o predicado P aplicado a todos os elementos da lista for verdadeiro, o predicado terceiro é verdadeiro

2.
Relativamente às afirmações apresentadas em seguida, diga se são verdadeiras ou falsas. No caso de serem falsas, justifique.

a) "Nas redes semânticas, a relação A é sub-tipo de B é representada por A⊆B."
Justificação:
Esta questao e verdadeira, pois o simbolo significa contido ou igual, no contexto da pergunta um subtipo pode ser exactmente igual logo e verdadeiro, o lsl disse que a quando de fazer a questao queria por (igual ou contido) mas nao tinha o caracter...

b) "Na pesquisa por recozimento simulado, à medida que o tempo passa, a pesquisa arrisca cada vez mais quanto a aceitar alterações com ganho negativo"
Justificação:
Falso
Quando o valor da função no nó actual é superior ao valor da função no sucessor, o sucessor é aceite com uma probabilidade que diminui exponencialmente em função da perda na função de avaliação.
Á medida que o tempo passa, a pesquisa arrisca cada vez menos quanto a aceitar alterações com ganho negativo

c) "Um programa é composto por um conjunto de um ou mais módulos e cada módulo tem uma função principal com o nome main."
Justificação:

Falso, por definição um módulo é isso mesmo, um ficheiro com predicados ou funções que são importados para uso para outro programa, sem main. Só o programa principal é suposto ter o **predicado** main.

d) "Na rede semântica representada pela lista [membro(a,b), relacao(a,c)] pode-se inferir o facto relacao(b,c)."
Justificação:

Falso. se A é membro de B, A é uma instância de B logo herda todas as propriedades do tipo a que pertence, entao apenas poderiamos inferir relação(b,c) se tivessemos [membro(b,a),relacao(a,c)].

e) "Um robô móvel autónomo pode ser considerado um agente"
Justificação:

Verdadeiro, desde que assumamos que o "robô móvel autónomo" tenha capacidade de obter informação sobre o seu ambiente (através de "sensores") e de executar acções em função dessa informação (através de "actuadores").
• Agente – uma entidade com capacidade de obter informação sobre o seu ambiente (através de "sensores") e de executar acções em função dessa informação (através de "actuadores").
• Exemplos:
– Agente físico: robô anfitrião
– Agente de software: agente móvel de pesquisa de informação na internet

f) "Na pesquisa por propagação de restrições, as restrições unárias não podem ser consideradas"
Justificação:
Falso, as restrições unitárias podem ser satisfeitas através de pré-processamento do domínio de valores da variável – aproveitam-se apenas os valores que satisfazem a restrição

g) "As funções em Mercury apenas podem manipular variáveis locais, ao passo que os predicados podem também manipular variáveis globais."
Justificação:
Falso, só há variáveis locais

3. Pretende-se elaborar um programa em Mercury para simular o comportamento das formigas na sua tarefa de arrumar provisões no formigueiro. A formiga procura provisões (acção procurar_provisão). Quando encontra uma provisão, agarra-a (acção agarrar_provisão) e vai procurar o local (acção procurar_local) de arrumação das provisões. A formiga tem sempre uma noção da distância percorrida desde que começou a procurar a arrumação. Se a formiga acha que já percorreu mais de 5 metros sem ter encontrado a arrumação, e vê outra formiga, vai atrás dela (acção seguir_formiga). Quando encontra o local onde estão as outras provisões, liberta a provisão que trás consigo (acção libertar_provisão). Cabe-lhe a si implementar um conjunto de regras situação-acção com base nas quais a formiga simulada se irá comportar.

a) Identifique e/ou defina em Mercury os tipos de dados a utilizar.

```
:-module formigas.

:-interface.

:-type condition ---> nao_tem_provisao ;
          tem_provisao ;
          distancia_percorrida(int) ;
          formiga_em_frente ;
          provisao_em_frente ;
          no_local.


:-type action ---> procurar_provisao ;
        procurar_local ;
        agarrar_provisao ;
```

```
        seguir_formiga ;
        libertar_provisao.

:-type state == set(condition).

:-func operator(action)= {set(condition),set(condition),set(condition)}.

:-pred transition(state::in,state::out,action::out) is nondet.

transition(State,NextState,procurar_provisao):-
   set.member(nao_tem_provisao,State),
   not(set.member(provisao_em_frente)),
   NextState = applyoperator(State,procurar_provisao).

transition(State,NextState,agarrar_provisao):-
   set.member(provisao_em_frente,State),
   set.member(nao_tem_provisao,State),
   NextState = applyoperator(State,agarrar_provisao).

transition(State,NextState,procurar_local):-
   set.member(tem_provisao,State),
   set.member(distancia_percorrida(N)), N<6,
   NextState = applyoperator(State,procurar_local).

transition(State,NextState,seguir_formiga):-
   set.member(formiga_em_frente,State),
   set.member(tem_provisao,State),
   set.member(distancia_percorrida(N),State), N>5,
   NextState = applyoperator(State,seguir_formiga).

transition(State,NextState,libertar_provisao):-
   set.member(no_local,State),
   set.member(tem_provisao,State),
   NextState = applyoperator(State,libertar_provisao).

operator(procurar_provisao) = {
      set.from_list([nao_tem_provisao]),
      set.from_list([nao_tem_provisao]),
      set.from_list([nao_tem_provisao])
      }.

operator(agarrar_provisao) = {
      set.from_list([provisao_em_frente, nao_tem_provisao]),
      set.from_list([provisao_em_frente, nao_tem_provisao]),
      set.from_list([tem_provisao])
      }.

operator(procurar_local) = {
         set.from_list([tem_provisao, distancia_percorrida(N)]),
         set.from_list([distancia_percorrida(N)]),
         set.from_list([distancia_percorrida(N1)])
         }:- N<6, N1=N+1.

operator(seguir_formiga) = {
         set.from_list([formiga_em_frente, tem_provisao, distancia_percorrida(N)]),
         set.from_list([distancia_percorrida(N)]),
         set.from_list([distancia_percorrida(0)])
         } :- N>5.

operator(libertar_provisao) = {
```

```
        set.from_list([no_local, tem_provisao]),
        set.from_list([tem_provisao]),
        set.from_list([nao_tem_provisao])
        }.

applyoperator(S, A) = R :-
   operator(A) = {_, N, P},
   R = set.union(set.difference(S, N), P).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % `set.list_to_set(List, Set)' is true iff `Set' is the set
        % containing only the members of `List'.
        %
%     :- pred set.list_to_set(list(T)::in, set(T)::out) is det.
%     :- func set.list_to_set(list(T)) = set(T).

        % Synonyms for set.list_to_set/1.
        %
%     :- func set.from_list(list(T)) = set(T).


 % `set_union(SetA, SetB, Set)' is true iff `Set' is the union of
        % `SetA' and `SetB'.  If the sets are known to be of different
        % sizes, then for efficiency make `SetA' the larger of the two.
        % (The current implementation using sorted lists with duplicates
        % removed is not sensitive to the ordering of the input arguments,
        % but other set implementations may be, so observing this convention
        % will make it less likely that you will encounter problems if
        % the implementation is changed.)
        %
%     :- pred set.union(set(T)::in, set(T)::in, set(T)::out) is det.
%     :- func set.union(set(T), set(T)) = set(T).

        % `set.difference(SetA, SetB, Set)' is true iff `Set' is the
        % set containing all the elements of `SetA' except those that
        % occur in `SetB'.
        %
%     :- pred set.difference(set(T)::in, set(T)::in, set(T)::out) is det.
%     :- func set.difference(set(T), set(T)) = set(T).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.
a) Enquadre a linguagem KIF no contexto da engenharia do conhecimento, comparando-a com
outros formalismos seus conhecidos e comentando a sua relevância para a construção de agentes.

Engenharia do conhecimento é uma área da engenharia que envolve integrar o conhecimento humano em sistemas computacionais com vista a estes poderem resolver problemas que normalmente requerem um alto nível de conhecimento/experiência de um ser humano.

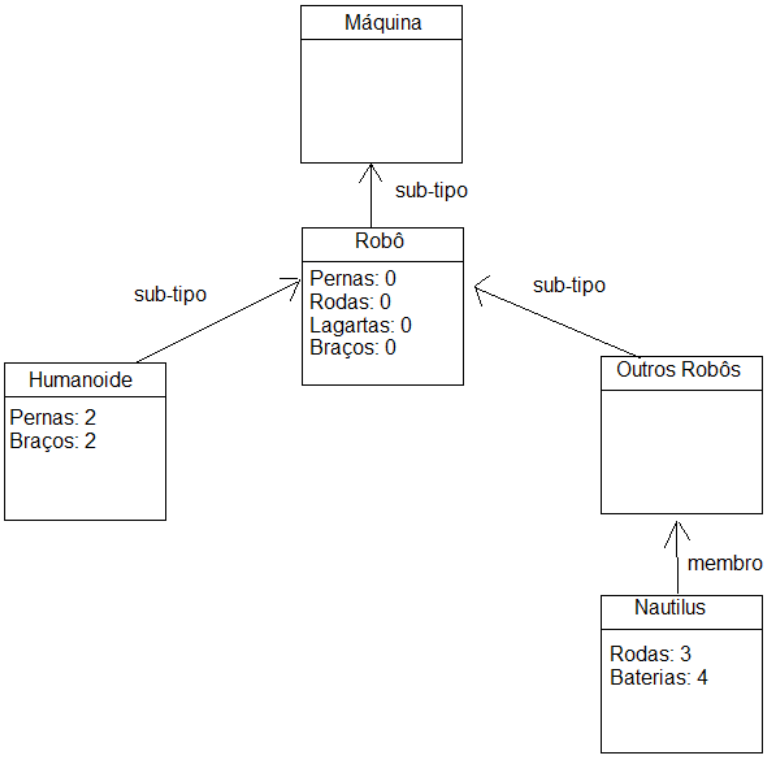
No presente, refere-se ao acto de construir, manter e desenvolver sistemas baseados no conhecimento. Tem muito em comum com a engenharia de software e é usada em áreas das Ciências da Computação como a Inteligência Artificial, entre outras.

Assim sendo, a KIF (*Knowledge Interchange Format*) pode-se considerar como a base comum da construção de agentes, visto ser um standard *de facto*. O uso do KIF permite a transferência da

informação de KIF para outros sistemas usados eventualmente pelos agentes, da mesma maneira que o Postscript foi um *standard* que permitiu o uso universal de um formato com qualquer modelo de impressora.

• Esta é uma linguagem desenhada para representar o conhecimento trocado entre agentes.
– A motivação para a criação do KIF é similar à que deu origem a outros formatos de representação, como o PostScript.
• Pode ser usada também para representar os modelos internos de cada agente.
• Características principais:
– Semântica puramente declarativa (o Prolog é também uma linguagem "comparando-a com outros formalismos seus conhecidos"
declarativa, mas a semântica depende em parte do modelo de inferência)
– Pode ser tão ou mais expressiva quanto a lógica de primeira ordem.
– Permite a representação de meta-conhecimento (ou seja, conhecimento sobre o conhecimento) para a construção de agentes"

b) Represente o seguinte conhecimento através de uma rede semântica: "Os robôs são máquinas. Há
robôs com pernas, que podem ou não ser humanóides, e robôs que se movem sobre rodas ou até usando lagartas. O
Nautilus é um robô com 3 rodas que obtém energia de 4 baterias de 12V / 7Ah. Os robôs humanóides têm 2
pernas e 2 braços."

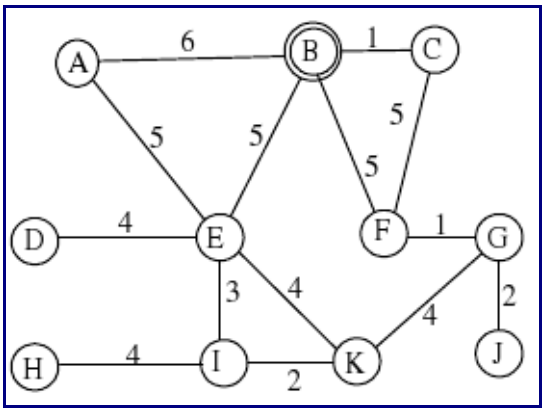


c) Considere a rede de Bayes identificada pela seguinte atribuição de probabilidades: p(a) = 0.2,
p(b|a) = 0.3, p(b|~a) = 0.2, p(c|b) = 0.2, p(c|~b) = 0.9, p(d|b) = 0.1, p(d|~b) = 0.2. Calculde a
probabilidade conjunta p(a ∧ b ∧ ~c ∧ ~d).

P(a ^b^~c^~d)=p(a)*p(b|a)*p(~c|b)*p(~d|b)
p(a)=0.2
p(b|a)=0.3

p(~c|b)=1-p(c|b)=0.8
p(~d|b)=1-(d|b)=0.9
P(a^b^~c^~d)=0.2*0.3*0.8*0.9=0.0432

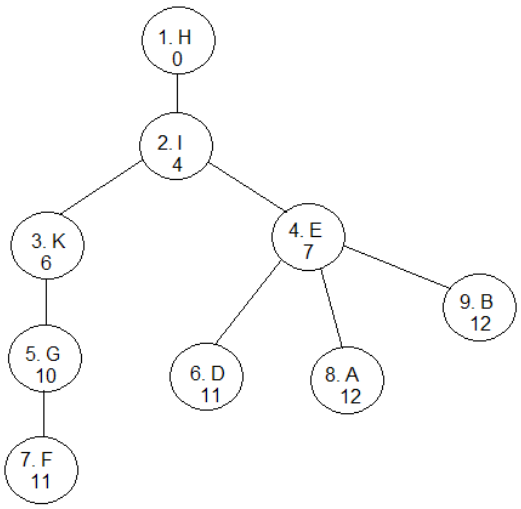
5. Considere que o grafo a seguir apresentado representa um espaço de estados num problema
de pesquisa. Os custos das transições estão anotados junto às ligações do grafo.



a) Tomando o estado H como estado inicial e o estado B como solução, apresente a árvore de pesquisa gerada, quando se realiza uma pesquisa de custo uniforme sem repetição de estados.
Numere os nós pela ordem em que são acrescentados à árvore e anote também o custo associado a cada nó.

1-> H custo=0 fila=[I/4]
2-> I custo=4 fila=[K/6,E/7]
3-> K custo=6 fila=[E/7,G/10]
4-> E custo=7 fila=[G/10,D/11,A/12,B/12] (pinhão)falei com o professor mariano e ele disse que aqui devia ficar [G/10,D/11,A/12,B/12,K/11,E/10] e assim sucessivamente
O Mariano tambem me respondeu ao mail agora a dizer o mesmo, no fundo vai dar ao mesmo, pq quando chega a vez do nó ser re-extraido da lista, é descartado pq ja foi visitado e a primeira solução é melhor
5-> G custo=10 fila=[D/11,F/11,K/11,A/12,B/12,J/12]
6-> D custo=11 fila=[F/11,K/11,A/12,B/12,J/12]
7-> K descartado
7-> F custo=11 fila=[A/12,B/12,J/12,C/16] também chega ao B com custo 16, mas B já está na fila com custo inferior
8-> A custo=12 fila=[B/12,J/12,C/16] também chega ao B com custo 18, mas B já está na fila com custo inferior
9-> B custo=12 fila=[J/12,C/16]

e pára aqui porque chegou à solução, eu sei que no link que pus o algoritmo não é assim, mas no que vi em mais sites é assim.

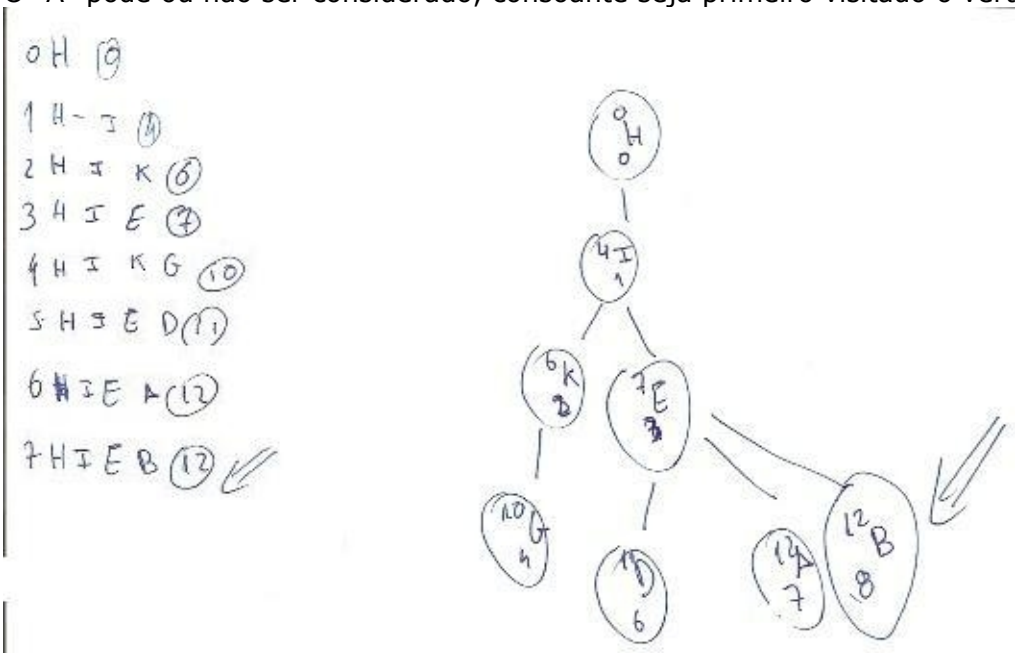


Entao ai quais sao os
N – número de nós da árvore de pesquisa no momento em que se encontra a solução
X – Número de nós expandidos (não terminais)
d – comprimento do caminho na árvore correspondente à solução

Davim:
N= 9
X= se eu tivesse de adivinhar dizia que eram 4, o E, o I, o K e o G // e o H não o consideram nao terminal pk?
d=12 (é a solução)

Nota: a ramificação média é um indicador da dificuldade do problema.
logo se o X for os nos expandidos (nao terminais da solução) quanto menor for maior é a ramificação media

<Manecas e Dr Lion com os cumprimentos do mariano>
O Pinhao foi ao gabinete do mariano hoje a tarde e uma das arvores possiveis de dar é esta:
O "A" pode ou não ser considerado, consoante seja primeiro visitado o vertice A ou B.



<Jin> não era suposto não se poder re-visitar nós?

Dr leaoooo - Ya esquecemo nos disso... e o mariano tb lol.(n corrigi a numeraçao na arvore o 6,7,8 sao 5,6,7
olha Mas o F no teu n ta a mais??
<Jin>foi o davim que fez, so fiz o desenho, mas concordo com ele pq o F se repararem tem custo 11, e os estado A e B tem 12

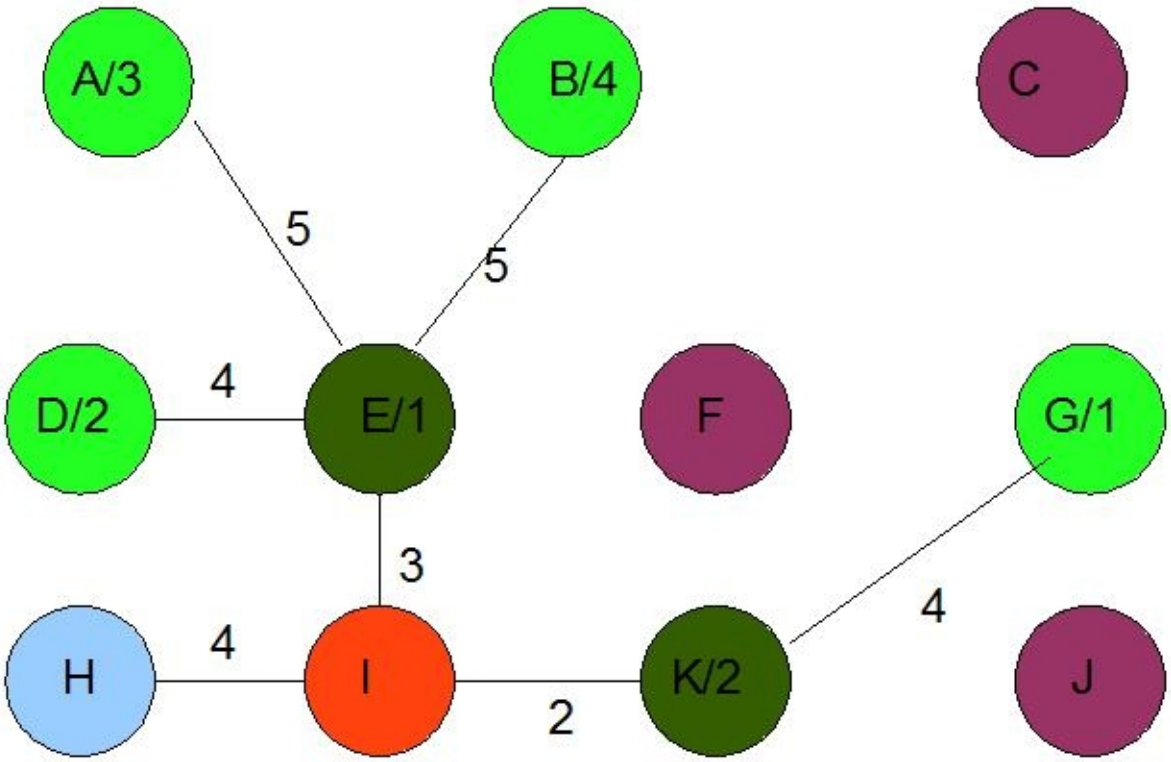
<Dr Leaoo> entao mas ele so exapande ate ao estado com custo 7 que é o estado E. Ao expandir o estado E encontra a solução.
<Jin> leao, mas o algoritmo procura em todos os nos que vai tirando da fila, e a fila é organizada do menor custo para o maior, ve a resoluçao do davim q entendes o q tou a dizer

Pesquisa em profundidade, é uma pesquisa não informada, por isso não chega a uma solução ideal, para qualquer coisa

H->I/4
H->I/4->K/6
H->I/4->K/6->E/10
H->I/4->K/6->E/10->D/14
H->I/4->K/6->E/10->D/14->A/15
H->I/4->K/6->E/10->D/14->A/15->B->21

E aparentemente para com esta solução, mesmo não sendo a melhor

Pesquisa em Largura:



A pesquisa em largura antes de avançar para o nível seguinte visita **todos** nós que estejam no nivel actual, por isso:

N=0-> azul claro
N=1-> laranja
N=2-> verde podre
N=3-> verde gay

Nota, está E/1 e K/2, mas deve ser K/1, e E/2, troquei quando estava a fazer a imagem,
peço desculpa se alguem andava confundido por causa disso

nós nao visitados estão a roxo
o valor que aparece nos nós é a ordem com que são visitados nesse nivel.
Mas provavelmente é irrelevante

Tão mas a solução do prof está bem ou n?
Porque é que tanto na pesquisa de profundidade como na de largura vão sempre primeiro ao K em vez do E. Nessas pesquisas o valor do custo das transições não conta para nada. Não deveria ser por ordem alfabetica, o E primeiro?

Davim: caso não diga não tenhas custos usas o critério que quiseres, mas quando tens custos nas arestas usa-los porque queres achar um caminho de custo minimo

Largura
H custo=0 profundidade=0
I custo=4 profundidade=1
E custo=7 profundidade=2
K custo=6 profundidade=2
A custo=12 profundidade=3
B custo=12 profundidade=3

Profundidade
H custo=0 profundidade=0
I custo=4 profundidade=1
E custo=7 profundidade=2
A custo=12 profundidade=3
B custo=12 profundidade=3

Não será assim?
E outra coisa se pesquisa em profundidade com limite crescente soma sempre +1 em vez do n inicial passa sempre a ser igual á pesquisa em largura a partir de profundidade inicial?
****
Olá,

O limite aumenta de 1 em 1. A pesquisa em profundidade com limite crescente usa uma fila Last-In-First-Out para guardar os nós por expandir. Logo não é igual à pesquisa em largura.
--
Pedro Mariano
****

b) Indique um valor aproximado do factor de ramificação efectivo da árvore gerada.

Davim:
Aqui tenho uma duvida, é que nos acetatos está:

Seja:
– N – número de nós da árvore de pesquisa no momento em que se encontra a solução
– X – Número de nós expandidos (não terminais)

– d – comprimento do caminho na árvore correspondente à solução na

o problema é que na pesquisa de custo uniforme sem repetição de estados ele continua sempre à procura de soluções até provar que a que tem é a melhor.
No exercicio, quando se acha a solução a árvore tem 4 nós, mas depois do algoritmo terminar tem 8... Qual é o N? tenho para mim se será 8... mas o professor
podia (devia) explicar-se melhor

<Jin> este vai ser aquele que ninguém resolve =X já na resoluçao do ano passado este não tava feito. Podiam perguntar ao mariano cm se faz =P

Davim: Eu perguntei:
como calculamos o factor de ramificação efectivo se nos slides diz que a expressão se resolve por métodos numéricos

Calculas um valor aproximado.

Portanto inventamos... ou entao levamos o portatil com matlab

blightcutter:
(
E assim rapaziada aplicando a formula N=B.((B^(d+1)-1)/(b-1) e tendo como d=3, pois esse e o nivel onde B (solução) aparece pela primeira vez, e N=11(numero de nos total) temos que por aproximação:

(nao vou por aqui os resultados):

2 e demasiado alto
1 valor invalido, da zero
3/2 muito alto
4/3 muito baixo
1.4 baixo
1.45 perfect... 11.05 blah

E assim disparem um valor acima (unidades) e um abaixo(unidades) depois vão aproximando....

Se quizerem calcular a ramificação media usem a formula (n-1)/x, onde x e o numero de nos expandidos (nao terminais) o D por exemplo seria um no terminal...

//////////////////////James
Tão mas não é da arvore gerada? O N não é 9 em vez de 11?
********** xavier
sim é
****