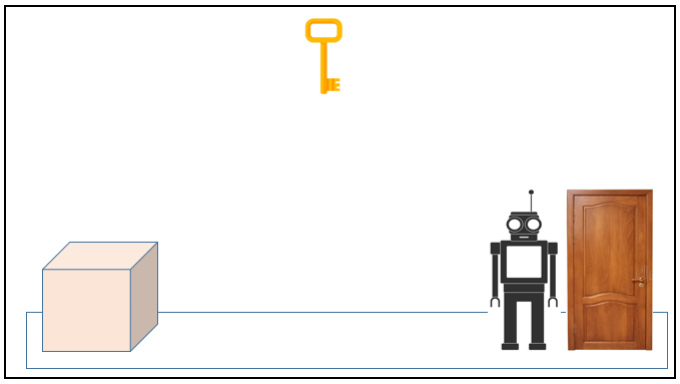
ACH2024 ALGORITMOS E ESTRUTURAS DE DADOS II

Semestre 2020-1 - Exercício prático – Percurso e robô– t04

Estagiário PAE: Samuel Caetano da Silva (samuel.caetano.silva@usp.br)

1. Considere o problema de um robô que precisa alcançar uma chave subindo em uma caixa, e usá-la para abrir uma porta de saída de um determinado ambiente com base em um número restrito de operações possíveis (subir, mover caixa para o lado, pegar chave etc.)



2. O ambiente é representado por uma cena vista do mesmo nível do robô (como se você estivesse observando por uma janela) de 3 x 3 quadros com linhas e colunas numerados de 1 a 3 a partir do topo e da esquerda (ou seja, a linha 1 é o teto e a linha 3 é o chão). O ambiente sempre contém um robô (bot), uma chave (key), uma caixa (box) e uma saída (exit). A seguir é apresentado um exemplo de configuração, dentre muitas alternativas possíveis.

	1	2	3	
1		key		
2				
3	box		bot	exit

3. Os componentes do ambiente são os seguintes:
   - o próprio robô (bot), que sempre começa em uma posição válida (junto ao chão na linha 3 ou em cima da caixa na linha 2, mas nunca está "voando").
   - a caixa (box) sempre no chão (linha 3), podendo inclusive ocupar a mesma posição do robô.
   - uma porta de saída (exit) sempre na extrema direita/esquerda. A saída pode estar no nível do chão (linha 3) ou da linha intermediária (2), neste caso exigindo que o robô suba na caixa para sair.
   - uma chave (key) que abre a porta, sempre na linha superior (1) e exigindo subir na caixa (na posição correta) para ser alcançada.
4. Um estado inicial neste sistema é definido pela sétupla representando as coordenadas do robô e da saída, as colunas da caixa e chave, e uma variável booleana indicando se o robô possui a chave ou não.

   estado(bot_lin, bot_col, exit_lin, exit_col, box_col, key_col, haskey)

5. Inicialmente todos os componentes da cena estão em posições válidas conforme as regras acima, e **haskey** é definido como *false*.
6. Para atingir o objetivo (i.e., sair da sala), o robô conta com oito ações possíveis indicadas por uma letra (r,l,u,d,L,R,g,e). Estas ações são listadas na tabela a seguir, juntamente com as regras (pré-condições) para que elas possam ser executadas, e os efeitos que sua execução teria sobre o estado do sistema.

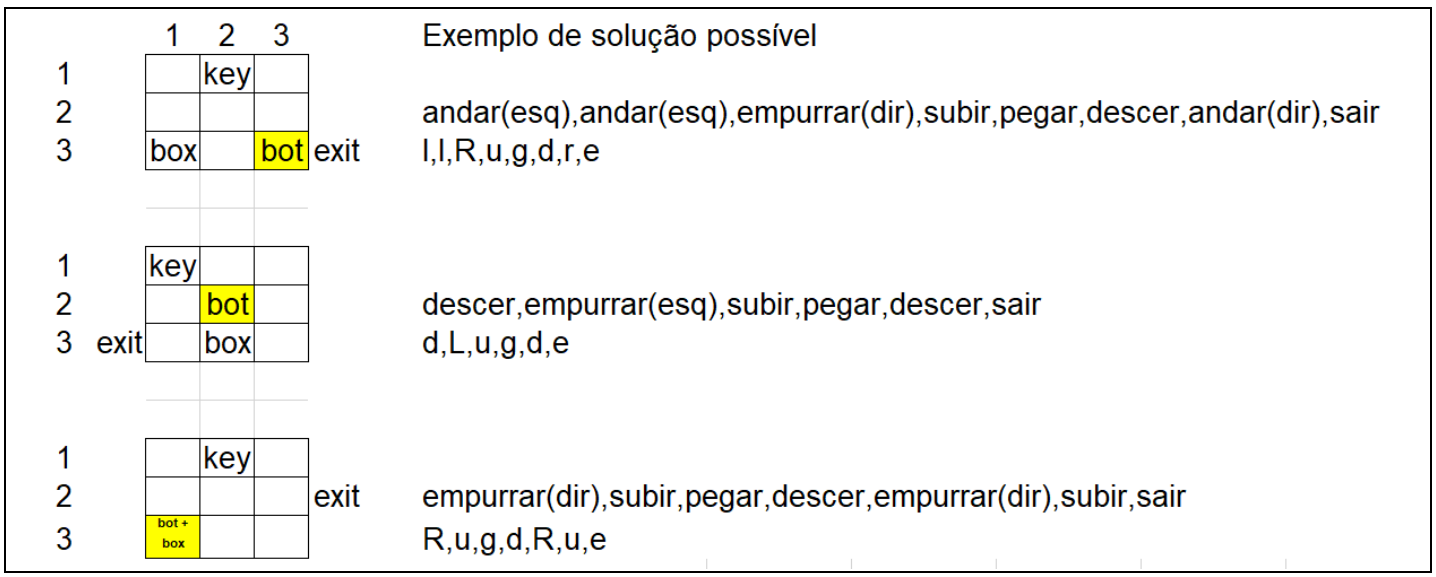
Ação	Descrição	Pré-condições	Efeitos
l (left)	Andar p/ esquerda	bot_col > 1	bot_col-- se bot_lin==2 então bot_lin=3 (robô cai)
r (right)	Andar p/ direita	bot_col < 3	bot_col++ se bot_lin==2 então bot_lin=3 (robô cai)
u (up)	Subir na caixa	bot_lin==3 bot_col==box_col	bot_lin--
d (down)	Descer da caixa	bot_lin==2	bot_lin=3
L (push left)	Empurrar a caixa p/esquerda e deslocar-se	bot_lin==3 bot_col==box_col bot_col>1	bot_col-- box_col--
R (push right)	Empurrar a caixa p/direita e deslocar-se	bot_lin==3 bot_col==box_col bot_col<3	bot_col++ box_col++
g (grab)	Pegar a chave	bot_lin==2 bot_col==key_col	haskey=True key_col=0 (remove chave)
e (exit)	Sair da sala	bot_lin==exit_lin bot_col==exit_col haskey==True	FIM (sucesso)

7. O objetivo do trabalho é implementar de forma correta e completa a função *caminhoValido* conforme modelo fornecido. A assinatura da função é a seguinte:

void caminhoValido(ESTADO *s, char* resp)

8. A função recebe como entrada um estado inicial *s e atualiza o vetor de caracteres *resp com uma sequência válida de ações que levariam o robô do estado inicial fornecido até a condição de sucesso. Note que há infinitas soluções válidas possíveis (por exemplo, o robô pode ir e voltar muitas vezes antes de decidir o que fazer), e que qualquer solução é válida desde que seja consistente com as regras do domínio.

9. A seguir são apresentados três exemplos de cenários e uma solução possível para cada um, em pseudocódigo e no formato string de caracteres que é a resposta real da função (**mas não inclua vírgulas no string de resposta**). As soluções deste exemplo não apresentam redundância (ou seja, não possuem movimentos inúteis), mas isso seria permitido.



10. Restrições de implementação:
    - Sua solução deve necessariamente contemplar o uso de um **algoritmo de busca em grafos** implementados em listas de adjacências ou em matrizes de adjacências, acompanhado das respectivas declarações *typedef* utilizadas.
    - Não defina variáveis globais.
    - Não exiba nenhuma mensagem na tela, nem solicite que o usuário pressione nenhuma tecla etc.
11. O EP pode ser desenvolvido individualmente ou em duplas, desde que estas sejam cadastradas até **sexta 27 de março** no link a seguir. Alunos que queiram trabalhar individualmente também devem se cadastrar, e duplas formadas tardiamente não serão consideradas. Por favor acesse o link abaixo usando seu email USP (solicitações de acesso com outros endereços serão ignoradas):

    https://docs.google.com/spreadsheets/d/1_wy0Uko3vX8GE07G1WP4X6X6-yRdu6eCpLdmmRvU7EM/edit?usp=sharing

    Importante: anote o número do seu grupo para informá-lo no código a ser entregue.
12. A função implementada deve estar inteiramente contida em um arquivo trabalho.cpp, incluindo todo código necessário para executá-la, declarações *typedef* etc.. Note no entanto que para testar a sua implementação e garantir sua correção você provavelmente terá de criar várias outras funções auxiliares (e.g., entrada de dados, exibição etc.) que não serão avaliadas. E que não devem ser entregues

O que/como entregar:

- 0 programa deve ser compilável no Codeblocks 13.12 sob Windows 7 ou superior. Será aplicado um **desconto de 50%** na nota do EP caso ele não seja **prontamente** compilável nesta configuração.
- Entregue um arquivo trabalho.cpp (exatamente com este nome, sem compactação) contendo a função do EP e todas as rotinas que ela invoca, inclusive as declarações *typedef* etc.
- A entrega será via *upload* no sistema Tidia por **apenas um** integrante de cada dupla.
- Preencha no código as declarações *nroUSP1*, *nroUSP2* e *grupo* para que você seja identificado. Se o EP for individual, mantenha o valor do segundo nro. como zeros.

Prazos:
O EP deve ser depositado no prazo definido na atividade cadastrada no sistema Tidia. Não serão aceitos EPs entregues depois do prazo ou incompletos, independentemente do motivo. Entregas no último dia são assim por conta e risco do aluno, e nenhum tipo de imprevisto de última hora (e.g., problemas de saúde, indisponibilidade de rede etc.) pode ser usado como justificativa para o atraso. O EP é uma atividade para ser desenvolvida ao longo de várias semanas, não no último dia da entrega.

O sistema Tidia envia um email de confirmação da submissão. É responsabilidade do aluno que fez o *upload* do arquivo verificar se o mesmo foi corretamente recebido pelo sistema (ou seja, sugere-se fazer *download* novamente para ter certeza). Atrasos/falhas na submissão invalidam o trabalho realizado, assim como entregas de versões erradas ou incompletas. Certifique-se também de que a versão entregue é a correta *antes* do prazo final. Não serão aceitas substituições.

Sugestão de implementação:
A partir do estado inicial fornecido, criar um grafo com vértices representando todos os estados possíveis, e arestas representando as operações de transição de um estado para outro. Feito isso, basta realizar uma busca pelo vértice (estado) final a partir do vértice inicial fornecido.

Avaliação:

O objetivo do exercício é o de gerar sequências de ações **corretas**, que não necessariamente precisam ser eficientes ou "elegantes". Ou seja, basta que o programa forneça uma resposta correta qualquer para cada estado de teste fornecido. Não existe assim solução "meio" correta: ou a função faz o que foi proposto, ou não faz. Erros de execução e de alocação de memória (muito comuns!) também invalidam o teste, assim como a ausência de eventuais funções auxiliares necessárias para a execução do programa.

O EP será testado com uma série de chamadas fornecendo-se sempre estados iniciais válidos como entrada. A porção do vetor de resposta que será avaliada vai da primeira posição (0) até a primeira ocorrência de uma letra "e" (indicando operação de sair). As posições do vetor após esta posição serão desprezadas, e se o vetor não possui nenhuma letra "e" a solução é considerada automaticamente inválida (inclusive porque isso iria ocasionar erro de execução).

Cada teste sem erro de execução e que termine na execução da ação "e" sem violar as regras do domínio vale um ponto. Para qualquer resultado diferente do esperado, passa-se ao próximo teste sem pontuar.

Este EP deve ser desenvolvido obrigatoriamente por *todos* os alunos de AED2. Sua nota é parte integrante da média final da disciplina e *não é* passível de substituição. Problemas com EPs – principalmente erros de execução e plágio – são a principal causa de reprovação na disciplina. Não tente emprestar sua implementação para outros colegas, nem copiar deles, pois isso invalida o trabalho de todos os envolvidos.

Não está funcionando? Use comandos *printf* para saber por onde seu programa está passando e os valores atuais das variáveis, ou os recursos de *debug* do *CodeBlocs*. O propósito do exercício é justamente o de encontrar erros por conta própria – não espere ajuda para isso. Bom trabalho!

Adaptado de Bratko, I. (1986) "Prolog: Programming for Artificial Intelligene." Addison-Wesley.