Guião da aula 9

Laboratório de Algoritmia I

Laboratórios de Informática II

Ano letivo 2019/20

Last Update: 04/03/2020 18:32:23

Tarefas a executar esta semana

As seguintes tarefas deverão ser entregues (através do github) até ao dia 19 de Abril às 20h00:

- Criar um módulo de listas ligadas genérico
- Criar o comando `jog`

Criação de um módulo de listas ligadas genérico

É **obrigatório** criar um módulo de listas ligadas. Ele **deve implementar** as funções com os seguintes protótipos:

```
LISTA criar_lista();
LISTA insere_cabeca(LISTA L, void *valor);
void *devolve_cabeca(LISTA L);
LISTA proximo(LISTA L);
LISTA remove_cabeca(LISTA L);
int lista_esta_vazia(LISTA L);
```

O tipo LISTA deve ser um apontador para uma estrutura. Eis um exemplo da utilização deste módulo:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "lista.h"
#define BUF_SIZE 1024

int main() {
  LISTA L = criar_lista();
  char linha[BUF_SIZE];

  printf("Insira várias linhas, acabando com CTRL-D:\n");

  // control-D é a tecla CTRL e a tecla D ao mesmo tempo
  // Em windows é capaz de ser CTRL-Z
    while(fgets(linha, BUF_SIZE, stdin) != 0) {
        // A função strdup cria uma cópia da string que foi lida
        L = insere_cabeca(L, strdup(linha));
    }

  printf("\n==============================\n");
  printf(  "=          PERCURSO          =\n");
  printf(  "==============================\n\n");
    // percorre sem remover os elementos da lista
    for(LISTA T = L; !lista_esta_vazia(T); T = proximo(T)) {
        char *str = (char *) devolve_cabeca(T);
        printf("%s", str);
    }
```

```
  printf("\n============================\n");
  printf(  "=          REMOCAO         =\n");
  printf(  "============================\n\n");
    // percorre e vai removendo a cabeça
    while(!lista_esta_vazia(L)) {
        char *str = (char *) devolve_cabeca(L);
        L = remove_cabeca(L);
        printf("%s", str);
        free(str);
    }
  return 0;
}
```

Informação sobre o novo comando

O comando `jog` deve permitir que o jogador atual peça ao computador para jogar por si. Nesta etapa, pretende-se que o comando funcione da seguinte forma:

1. Varrer todas as posições vizinhas da peça branca que estejam livres e armazená-las numa **lista ligada** de posições
2. Usar uma **heurística** para escolher qual é a jogada que vai ser efetuada
3. Fazer essa jogada e mudar o jogador atual

Heurística a implementar

Esta secção apresenta várias heurísticas possíveis. Para este guião, qualquer das heurísticas apresentadas abaixo será aceite. Para o guião 10, terão que implementar uma das outras estratégias.

Considerações iniciais

Sabemos que no jogo do Rastos um jogador ganha se:

1. Alguém chega à sua casa destino (neste caso não interessa qual foi o jogador que lá chegou)
2. O jogador acabou de jogar de tal maneira que o adversário não tem nenhuma jogada válida

Assim, as estratégias para jogar baseiam-se nestes dois objetivos. Como o primeiro objetivo pode não ser possível de conseguir, é preciso ter sempre em mente o segundo objetivo.

Possíveis estratégias

1. Escolha aleatória
2. Distância menor usando a distância Euclidiana
3. Distância menor usando o algoritmo Flood Fill
4. Estratégia baseada na paridade
5. Minimax
6. Monte Carlo Tree Search

Escolha aleatória

Neste caso, a heurística será escolher simplesmente uma das hipóteses possíveis.

Distância menor usando a distância Euclidiana

Neste caso pretende-se escolher a casa que fique mais perto do objetivo do jogador atual. Assim:

1. se o jogador atual for o jogador 1 deve-se escolher a casa mais perto em linha reta do canto inferior esquerdo (casa **1**)
2. se o jogador atual for o jogador 2 deve-se escolher a casa mais perto em linha reta do canto superior direito (casa **2**)

Distância menor usando o algoritmo Flood Fill

A heurística anterior não funciona corretamente caso hajam casas ocupadas, visto que a branca não pode ir para cima das casas que estão ocupadas. Neste caso, pretende-se usar o algoritmo *Flood Fill* para decidir qual é a casa que está mais próxima do destino. É possível que não haja caminho para a casas destino; neste caso, sugere-se que:

1. jogue aleatoriamente, ou
2. jogue para a posição que lhe seja proveitosa para ganhar por não deixar nenhuma jogada ao adversário.

Estratégia baseada na paridade

Esta estratégia conta as áreas e considera que se no fim de jogar deixarmos uma área com um número par de casas livres, então ganharemos desde que todas essas casas sejam preenchidas sistemáticamente. Assim, esta estratégia conta, para cada jogada possível, o tamanho da área que se deixa para o outro jogador e escolhe aquela que deixa uma área com um tamanho par.

Algoritmo de procura Minimax

Este algoritmo é utilizado quando queremos prever o que vai acontecer daqui a várias jogadas. Neste caso, o que se pretende é criar uma função (mais uma vez uma heurística) que avalie cada posição. Após isso, o algoritmo irá experimentar todas as jogadas possíveis de cada um dos jogadores, até uma dada profundidade e avaliar a posição final em cada caso e escolher a jogada mais proveitosa para o jogador atual. Assume-se que cada jogador escolhe a jogada mais proveitosa. A razão do algoritmo se chamar *Minimax* é que enquanto que o jogador atual tenta maximizar a sua posição, o adversário pretende minimizar essa vantagem.

Exemplo de utilização

Segue-se um exemplo em que se utilizou a estratégia da escolha aleatória:

```
8 .......2
7 ........
6 ........
5 ....*...
4 ........
3 ........
2 ........
1 1.......
  abcdefgh
# 01 PL1 (1)> e4
8 .......2
7 ........
6 ........
5 ....#...
4 ....*...
3 ........
2 ........
1 1.......
  abcdefgh
# 02 PL2 (1)> d5
8 .......2
7 ........
```

```
 6 ........
 5 ...*#...
 4 ....#...
 3 ........
 2 ........
 1 1.......
   abcdefgh
 # 03 PL1 (2)> jog
 8 .......2
 7 ........
 6 ........
 5 ...##...
 4 ..*.#...
 3 ........
 2 ........
 1 1.......
   abcdefgh
 # 04 PL2 (2)>
```