Estruturas lineares

março/2019

Exercício 1

- Utilizando a estrutura de pilhas, faça uma função que inverte uma string utilizando uma estrutura de pilha. O argumento “**s**” representa a string a ser invertida e o argumento “**dest**” uma string pré-alocada que conterá a string invertida.

```
void inverte_string(char *s, char *dest);
```

Exercício 2

- Uma expressão matemática pode ser escrita de forma:
  - Infixa: `2 * 3 + 4`
  - Prefixa: `+ * 2 3 4`
  - Pósfixa: `2 3 * 4 +`
- Para avaliar uma expressão pósfixa, seguimos o seguinte algoritmo:
  - Leia o próximo caractere '**c**';
  - Se '**c**' for um número, insere na pilha "**p**";
  - Se '**c**' for um dos operadores matemáticos, desempilha dois números da pilha, efetua a operação e empilha o resultado;
  - Se '**c**' for qualquer outro símbolo, ignore;
  - Caso ainda exista caracteres, volte ao primeiro passo;
  - Retorne como resultado o único valor que ficar na pilha após ler toda a expressão.
- Assumindo números de apenas um dígito, implemente um programa que avalie uma expressão pósfixa.

Exercício 3

- O problema das 8 rainhas consiste em colocar 8 rainhas em um tabuleiro de xadrez de tal forma que nenhum par de rainhas se ataque.
- Uma forma de encontrar uma solução para esse problema utiliza uma representação de array de 8 elementos, em que cada posição do vetor representa uma coluna do tabuleiro e o valor a linha correspondente em que a rainha se encontra. O valor **-1** indica que aquela rainha ainda não foi alocada.

```
int sol[8] = {1, 2, 3, 4, -1, -1, -1, -1}
```

Exercício 3

- Dado o estado:

```
int s0[8] = {-1, -1, -1, -1, -1, -1, -1, -1}
```

- Os próximos estados podem ser:

```
{1, -1, -1, -1, -1, -1, -1, -1}
{2, -1, -1, -1, -1, -1, -1, -1}
{3, -1, -1, -1, -1, -1, -1, -1}
{4, -1, -1, -1, -1, -1, -1, -1}
{5, -1, -1, -1, -1, -1, -1, -1}
{6, -1, -1, -1, -1, -1, -1, -1}
{7, -1, -1, -1, -1, -1, -1, -1}
{8, -1, -1, -1, -1, -1, -1, -1}
```

Exercício 3

- O algoritmo funciona da seguinte forma:
  1. Insere o estado “s0” em uma fila;
  2. Remove o primeiro elemento da fila em “s”;
  3. Para cada possível próximo estado de “s”, se for uma solução termina. Caso contrário, se for factível, insere na fila;
  4. Se a fila estiver vazia, termina com erro, senão retorna ao passo 2.

Exercício 3

- Para verificar se uma solução é factível utilize a função:

```
int ataca(int linha1, int col1, int linha2, int col2) {
    return linha1 == linha2 || col1 == col2
           || abs(linha1-linha2) == abs(col1-col2);
}

int factivel(int * sol) {
    int c1, c2;
    for (c1 = 0; c1 < 7; c1++) {
        if (sol[c1] == -1) return 1;
        for (c2 = c1+1; c2 < 8; c2++) {
            if (sol[c2] != -1)
                if (ataca(sol[c1], c1+1, sol[c2], c2+1)) return 0;
        }
    }
    return (sol[7] == -1 ? 1 : 2); // Retorna 2 para uma solucao
}
```