Fundamentos de Programação

Filipe Francisco

10 de maio de 2018

Problemas

- Problemas envolvendo matrizes:

- Leitura/preenchimento da matriz:
  - para ler um vetor, utilizávamos um loop para preencher cada posição
  - para uma matriz, precisamos preencher cada posição de cada linha
  - portanto, utilizaremos dois loops
    - um loop para a linha (por exemplo, um loop em a)
    - um loop para a coluna (por exemplo, um loop em b)
    - deste modo, dentro dos loops, vamos ler $M[a][b]$
- Bingo:
  - V1: recebe apenas uma lista de números sorteados
    - a cartela é a mesma, ou seja, vamos inicializar a matriz diretamente no código
  - V2: recebe a lista de números sorteados, e a cartela
    - a cartela varia, ou seja, vamos inicializar a matriz lendo valores

Problemas

- Problemas envolvendo matrizes:

- Soldados em posição:
  - comparamos um soldado de uma fila (linha da matriz), com um soldado uma fila atrás (ou seja, uma linha abaixo)
  - problema: a última linha da matriz não tem outra linha abaixo dela!
  - para resolver este problema, basta eliminarmos a verificação para a última linha da matriz
    - deste modo, ao chegar na linha 2, não vamos compará-la com a 3 (que não existe)
- Matriz simétrica:
  - matriz que é igual a sua transposta
  - ou seja: a linha da matriz é igual a sua coluna
    - linha 0 deve ser igual à coluna 0
    - linha 1 deve ser igual à coluna 1
    - linha 2 deve ser igual à coluna 2
  - portanto, basta verificarmos se todos os elementos das linhas são iguais aos das colunas!

Problemas

- Problemas envolvendo matrizes:

- Jogo da velha:
  - matriz com 'V' (vazio), 'O' e 'X'
  - leitura da matriz: pode ser caractere a caractere, ou em forma de string
  - para resolver este problema, basta verificarmos todos os casos possíveis de vitória
  - casos possíveis:
    - completar linha 0, 1, ou 2
    - completar coluna 0, 1, ou 2
    - completar diagonal principal
    - completar diagonal secundária
  - além disto, devemos eliminar o caso de termos preenchido por 'V'
    - exemplo: uma linha completa está preenchida por 'V'
    - isto não significa que alguém ganhou!

Funções

- Em uma aula anterior, falamos sobre funções
- Vimos que elas são úteis para reutilizar um mesmo código várias vezes
- Hoje, vamos comentar dois tópicos envolvendo funções
  - pilha de execução
  - função recursiva

Pilha de execução

- A pilha de execução guarda informações sobre as funções ativas na execução de um programa
  - pilha: trabalhamos com o que está no topo!
- Por exemplo, lembre-se do problema do Triângulo de Pascal
  - imprimir todas as linhas de 0 até um certo n
- A função TriânguloPascal chamava a função LinhaTriangulo, que chamava a função Binomial, que chamava a função Fatorial
- Após uma função chamar outra, temos que saber de onde devemos retomar cada função
- A pilha de execução existe para isto!
  - sabendo de onde devemos continuar a execução, podemos retomar a execução da função anterior exatamente de onde paramos!

Funções recursivas

- Ainda sobre funções, vimos que durante sua execução, uma função pode chamar outra
- Vimos alguns exemplos de como isso funciona
  - exemplo anterior: a função TriânguloPascal chamava a função LinhaTriangulo, que chamava a função Binomial, que chamava a função Fatorial
- O que não comentamos é que uma função também pode chamar a ela mesma!
  - ou seja, uma função *fun* chama *fun* no seu código
- Mas para que isto é útil?
- Antes de conversarmos mais, vamos dar um exemplo

Funções recursivas

- Suponha que você tem várias cartas numeradas e deseja ordená-las
- Você pode ordenar todas as cartas você mesmo
  - por exemplo, você pode aplicar um Bubble Sort manualmente
- Porém, você também pode fazer o seguinte:
  - dividir as cartas em dois blocos, e pedir para duas pessoas A e B ordenarem cada um dos blocos
  - após terem ordenado, você pega os dois blocos ordenados individualmente e os junta em um único bloco completo de cartas ordenadas
- E juntar dois blocos ordenados é fácil!
  - basta olhar as cartas do topo e escolher a menor delas
- Note que as pessoas A e B podem aplicar esta mesma ideia recursivamente
  - isto é, A divide o bloco para duas pessoas C e D, e B faz o mesmo para duas pessoas E e F, e as pessoas C, D, E, F podem fazer o mesmo

Funções recursivas

- Basicamente, o que apresentamos no slide anterior é uma função recursiva para ordenar uma sequência numérica
  - de fato, isto é um algoritmo de ordenação que vocês verão mais pra frente!
- A ideia de recursão (ou recorrência) é isso!
- E uma das melhores aplicações dela é exatamente trabalhar sobre parte do problema!
- Uma função recursiva é uma função que, para resolver o problema, faz uma chamada a ela mesma para parte do problema!
  - para ordenar n elementos, fazemos duas ordenações de $\frac{n}{2}$ elementos
- Funções recursivas muitas vezes são bem simples, e bem úteis!

Funções recursivas

Exemplo 1: Fatorial

- Problema: calcular o fatorial de um número
  - este é um problema básico que já vimos, e que sabemos resolver utilizando um loop
- Pela definição de fatorial, temos: $n! = n \cdot (n-1)!$
  - $(n-1)! = (n-1) \cdot (n-2) \cdot \cdots \cdot 3 \cdot 2 \cdot 1$
  - logo, $n \cdot (n-1)! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 3 \cdot 2 \cdot 1 = n!$
- Note que, desta forma acabamos de definir o fatorial recursivamente
  - o fatorial de n é igual a n vezes o fatorial de $n-1$
  - acabamos de definir fatorial em função de fatorial
- Problema: se simplesmente ficarmos diminuindo, alcançaremos números negativos

Funções recursivas

Exemplo 1: Fatorial

- Portanto, devemos definir um caso base!
  - caso base: caso (ou casos) mais simples possível, para o qual sabemos a resposta
  - no caso do fatorial, sabemos que $0! = 1$
- A partir disto, podemos identificar os casos para montarmos nossa função, e podemos montar uma equação de recorrência
  - ela define como funcionará a recursão
- Temos a seguinte equação de recorrência:

$$fat(n) = \begin{cases} 1, & \text{se } n = 0 \\ n \cdot fat(n-1), & \text{se } n > 0 \end{cases}$$

Funções recursivas

Exemplo 1: Fatorial

Algoritmo: FatorialRecursivo(n)

Entrada: inteiro n

Saída: valor do fatorial de n

1 **se** $n = 0$ **então retorne** -1

2 **retorne** $n \cdot$ FatorialRecursivo$(n - 1)$

Funções recursivas

Exemplo 2: Busca Binária

- Problema: procurar por um valor x em um vetor ordenado
- Ideia básica: percorrer o vetor do começo ao fim procurando por x
- Porém, temos uma maneira mais eficiente de fazermos isso!
  - suponha que temos um vetor V e estamos trabalhando de um índice inicial i a um índice final f
  - vamos calcular a posição m como $\frac{i+f}{2}$ (esta é a posição do meio entre i e f)
  - comparamos x e $V[m]$
    - se $x = V[m]$, encontramos!
    - se $x > V[m]$, então x estará depois de y no vetor (se existir), portanto devemos buscar na metade da direita
    - se $x < V[m]$, então x estará antes de y no vetor (se existir), portanto devemos buscar na metade da esquerda
  - as ideias de buscar na esquerda e na direita representam aplicações recursivas do método (ou seja, chamadas recursivas)

Funções recursivas

Exemplo 2: Busca Binária

- Se o elemento x estiver no vetor V, este método já encontrará o elemento
- Problema: e se x não estiver em V?
  - neste caso, acabaremos caindo em uma situação onde temos apenas um elemento restante
  - ao compararmos x e $V[m]$, faremos uma chamada recursiva para a metade esquerda (ou direita) do vetor
  - porém, só temos um elemento, não há metade!
  - logo, chamamos recursivamente para um vetor de tamanho zero!
    - vetor de tamanho maior que 1: $i < f$
    - vetor de tamanho 1: $i = f$
    - vetor de tamanho 0: $i > f$
- Portanto, só fazemos as verificações do slide anterior se tivermos pelo menos um elemento
  - ou seja, basta eliminar os casos com vetor vazio!

Funções recursivas

Exemplo 2: Busca Binária

Algoritmo: BuscaBinaria(V, i, f, x)

Entrada: vetor V, índices inicial i e final f, inteiro n

Saída: índice de x no vetor V, ou -1 se não existir x em V

1 **se** $i > f$ **então retorne** -1

2 $m = \frac{i+f}{2}$

3 **se** $x = V[m]$ **então retorne** m

4 **se** $x < V[m]$ **então retorne** BuscaBinaria($V, i, m-1, x$)

5 **se** $x > V[m]$ **então retorne** BuscaBinaria($V, m+1, f, x$)
