Avaliação 2

Daniel Taiki Ukita e Gustavo Veronica Santos

**Exercício 1):**

O problema consiste em inverter um número inteiro (N) de forma recursiva e retornar o resultado como um novo número inteiro. Como descrito no exemplo abaixo:

1234 / 10 = 123

1234 % 10 = 4

123 / 10 = 12

123 % 10 = 3

12 / 10 = 1

12 % 10 = 2

1 / 10 = 0

1% 10 = 1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |

x10000

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3 | 2 | 1 |

x100

4000 + 300 + 20 +1 = 4321

x10

A lógica do método inverso é baseada na recursão e na manipulação de dígitos. O método verifica se o número tem um único dígito e, se sim, retorna o próprio número. Para números maiores, ele decompõe o número em seus dígitos individuais, processa o último dígito e então chama a si mesmo recursivamente para processar o restante do número. Quando multiplicamos o último dígito de um número (N % 10) por um valor específico, conseguimos colocar esse dígito na posição correta de um novo número que estamos formando. Esse valor específico é obtido dividindo a variável ‘aux’ por 9. A variável ‘aux’ é um número grande o suficiente para que, quando dividido por 9, nos dê exatamente a posição onde o dígito deve ficar no número que está sendo invertido. Depois, usamos uma técnica chamada recursividade, que é como um loop, para repetir esse processo com os próximos dígitos do número original. Isso é feito pela função inverso(N / 10), que pega o número original sem o último dígito e repete o processo até que todos os dígitos sejam processados e o número esteja completamente invertido.

Observação: A variável aux é essencial no programa para posicionar corretamente os dígitos durante a inversão do número. Ela é configurada para ser pelo menos igual ou maior que o número N, sempre em um múltiplo de 10. Isso permite calcular a posição adequada dos dígitos no número resultante. A falta de aux impede a realocação dos dígitos, levando a uma inversão errada.

Tabela

Descrição gerada automaticamente**Exercício 2):**

Para a resolução do exercício, iniciamos identificando um padrão que possibilitasse a aplicação de um método recursivo. Observasse que a tabela apresenta uma característica distinta: o valor da primeira coluna é sempre 1. Este elemento constante serve como base para a construção dos valores subsequentes. À medida que avançamos para as colunas à direita, cada novo valor é calculado pela soma dos elementos acima e à esquerda, depositando o resultado na célula correspondente da linha seguinte. Como pode ser visto na tabela abaixo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lin/Col. | 1 | 2 | 3 | 4 | 5 |
| 1 | **1** | **0** | 0 | 0 | 0 |
| 2 | 1 | **1** | 0 | 0 | 0 |
| 3 | 1 | 2 | 1 | 0 | 0 |
| 4 | 1 | 3 | 3 | 1 | 0 |
| 5 | 1 | 4 | 6 | 4 | 1 |

Utilizando da técnica da recursão, a função tabela(lin, col) é estruturada para se chamar repetidamente em uma versão mais simples de si mesma. Em cada chamada recursiva, a função se desdobra em tabela(lin-1, col-1) e tabela(lin-1, col), que são, respectivamente, a célula diagonalmente acima à esquerda e a célula diretamente acima da posição atual. O problema também apresenta dois casos bases que permitem resolver o problema sem necessidade de recursividade. Primeiro caso: quando o número da coluna excede o da linha (ou seja, col > lin), o valor correspondente na tabela é 0, pois não existem elementos anteriores para somar. Segundo caso: na primeira coluna de qualquer linha (isto é, col == 1), o valor é sempre 1, formando a base estrutural da tabela.

O algoritmo que compõe a tabela pode se apresentar da seguinte forma:

1. Se (col > lin), retorne 0.
2. Se (col == 1), retorne 1.
3. Caso contrário, retorne tabela(lin-1, col-1) + tabela(lin-1, col).

**Exercício 3):**

Para a resolução do exercício, iniciamos identificando um padrão que possibilitasse a aplicação de um método recursivo. Por se tratar de uma lista encadeada, cada nó pode se tratar como uma estrutura independente, que compõe a lista. Cada nó tem um apontador para o próximo nó, formando uma subestrutura que é uma versão menor do problema original. Isso permite que a função recursiva prossiga para o próximo nó como uma nova instância do mesmo problema. Em relação ao caso base é alcançado quando o nó atual é nulo, indicando o fim da lista, já possuindo um critério de parada. O exemplo abaixo representa o funcionamento da lista:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | **3** | 4 | **3** | 4 | 5 | 7 | **3** |

Verificar se o número 3 aparece mais de 2 vezes na lista.

1 - Verifica se o nó atual é nulo. Se for, retorna o contador atual (que seria 0 no início).

2 - Compara o valor do nó atual 2 com o valor procurado 3.

3 - Como são diferentes, não incrementa o contador. Faz uma chamada recursiva para o próximo nó (com valor 3) e repete o processo.

4 - No segundo nó, o valor é 3, então incrementa o contador para 1 e faz uma chamada recursiva para o próximo nó.

5 – Ao fim do processo é impresso True, demostrando que o valor de 3 aparece mais de 2 vezes na lista encadeada.

Para identificar valores repetidos em uma lista encadeada, utilizamos um método chamado repete. A lógica é: se a quantidade de vezes que o valor N aparece na lista já excedeu o número desejado, o método retorna verdadeiro, pois o objetivo foi alcançado. Caso contrário, o método verifica se chegou ao final da lista. Se sim, e o valor N foi encontrado o número necessário de vezes, também retorna verdadeiro. Se ainda não chegou ao final, o método verifica o valor do nó atual. Se for igual a N, decrementa a quantidade e continua a busca no próximo nó. Se for diferente, apenas continua para o próximo nó sem alterar a quantidade. Esse processo se repete até que a condição seja satisfeita ou até que todos os nós sejam verificados.