Cálculo Fatorial Usando Programação Dinâmica (Top-Down e Bottom-Up)

Índice de Conteúdo:

Pseudocódigo

Função calcularFatorialTopDown(n):

```
Se n for igual a 0 ou 1:
  Retornar 1  // Condição de parada

Se memo[n] já estiver preenchido:
  Retornar memo[n]  // Usa valor já calculado

Caso contrário:
  memo[n] = n * calcularFatorialTopDown(n - 1)  // Chamada recursiva com armazenamento

Retornar memo[n]
```

Função calcularFatorialBottomUp(n):

```
Inicializar resultado = 1

Para i de 2 até n:
  resultado = resultado * i

Retornar resultado
```

Estrutura principal:

```
Início
  Inicializar array memo[0...n] (para Top-Down)

  Ler número n

  Se Top-Down:
    Chamar calcularFatorialTopDown(n)

  Se Bottom-Up:
    Chamar calcularFatorialBottomUp(n)

  Exibir o resultado

Fim
```

Abordagem Top-Down (Recursiva com Memoization):

A abordagem **Top-Down** usa recursão e memoization para armazenar resultados intermediários. Sempre que um subproblema é resolvido, ele é salvo em um array memo[] para evitar cálculos repetidos em chamadas futuras.

- **Recursão**: A função chama a si mesma para calcular o fatorial de números menores até chegar ao caso base (n = 0 ou n = 1).
- **Memoization**: Armazena os resultados intermediários em um array (memo[]) para evitar recalcular subproblemas repetidos.
- **Espaço**: Usa mais memória, pois precisa armazenar os resultados de todos os subproblemas em memo[] (O(n)).
- **Complexidade**: Pode sofrer de **estouro de pilha** se n for muito grande, devido ao número de chamadas recursivas.

Abordagem Bottom-Up (Iterativa):

Na abordagem **Bottom-Up**, o cálculo do fatorial é realizado iterativamente, de baixo para cima. Começa-se com o valor base do fatorial de 1 e, em seguida, multiplica-se progressivamente até o valor desejado de n.

- **Iteração**: Usa um loop para calcular o fatorial progressivamente, de 1 até n.
- **Armazenamento**: Não armazena subproblemas, apenas o valor acumulado do fatorial (O(1) em termos de espaço extra).
- **Espaço**: Usa menos memória, pois não precisa de estrutura adicional além de variáveis temporárias.
- **Estabilidade**: Não corre o risco de estouro de pilha, pois não utiliza recursão.

Explicação Geral:

Entrada:

O programa lê um número inteiro n e calcula seu fatorial, utilizando uma das duas abordagens: Top-Down ou Bottom-Up. A implementação lida com valores que variam de pequenos inteiros até números grandes (como 100!).

Condição de Parada:

Para ambos os métodos, a condição de parada ocorre quando n é 0 ou 1, retornando 1. Esse é o ponto em que o cálculo termina.

Saída:

O resultado final do fatorial é exibido no console. Para números muito grandes, o resultado pode ser exibido em notação científica para facilitar a leitura.

Manipulação de Números Muito Grandes:

Limitação de Precisão:

Para lidar com números grandes (como o fatorial de 100), utilizamos a classe BigInteger, que permite a manipulação de números de tamanho arbitrário. Números grandes demais são formatados para exibição com notação científica.

Função formatarResultadoFatorial:

- **Contagem de Dígitos**: Verifica o número de dígitos do resultado.
- **Notação de Potência de 10**: Se o número tiver mais de 10 dígitos, exibe o número em notação científica no formato 10^expoente.
- **Formatação Decimal**: Adiciona o valor completo em decimal abaixo da notação científica.

Exibição de Resultados:

Para valores grandes como n = 100, o fatorial será exibido da seguinte forma:

```
Fatorial de 100 é: 10^157

Em formato decimal completo:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000E157
```

Análise Assintótica:

Abordagem Top-Down (Recursiva com Memoization):

- **Complexidade Temporal**: O algoritmo resolve cada subproblema uma vez, resultando em uma complexidade de tempo de O(n).
- **Complexidade Espacial**: O uso de memória é O(n), pois cada subproblema é armazenado no array memo[].

Abordagem Bottom-Up (Iterativa):

- **Complexidade Temporal**: O loop iterativo executa n iterações, resultando em O(n) para o tempo de execução.
- **Complexidade Espacial**: O espaço adicional necessário é O(1), já que o cálculo é feito diretamente em variáveis temporárias.

Comparação entre Abordagens:

- **Top-Down**: O(n) para tempo e O(n) para espaço.
- **Bottom-Up**: O(n) para tempo e O(1) para espaço.

A abordagem **Bottom-Up** é mais eficiente em termos de uso de memória, enquanto a **Top-Down** com memoization pode ser mais intuitiva para problemas que têm subproblemas sobrepostos.

Observações:

- **Limitações da Recursão**: Para valores muito grandes de n, a abordagem recursiva (Top-Down) pode causar estouro de pilha (stack overflow). A abordagem iterativa (Bottom-Up) é mais estável nesses casos.
- **BigInteger**: É necessário para manipular números grandes. A conversão para notação científica é importante para manter a legibilidade dos resultados.