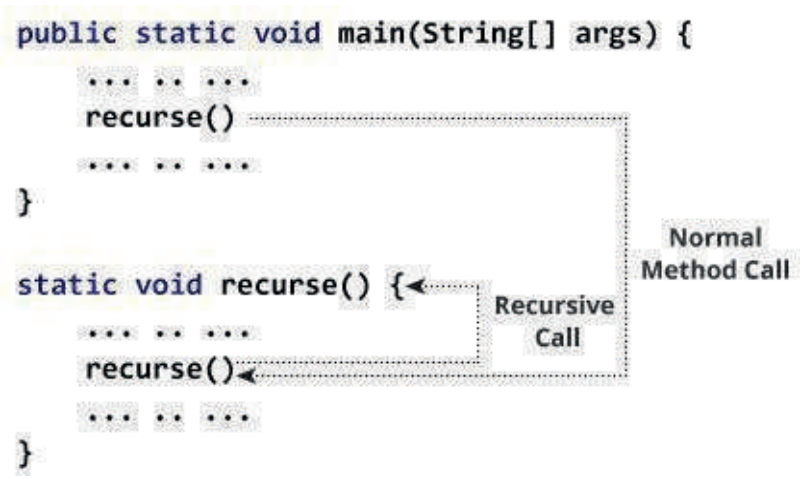
Google olabi

<RECURSÃO& PROGRAMAÇÃODINÂMICA>

{ Recursão

Recursão é a técnica de fazer uma função chamar por ela mesma. Esta técnica fornece uma maneira de simplificar problemas complicados de forma a torná-los mais fáceis de resolver.



Na chamada recursiva acima, chamamos o método **recurse()** de dentro do método **main.** Uma vez dentro do método **recurse()**, chamamos novamente o método **recurse**. Para interromper a chamada recursiva, precisamos fornecer algumas condições dentro do método, do contrário, o método será chamado infinitamente. Usamos então a instrução **if...else** (ou abordagem semelhante) para encerrar a chamada recursiva dentro do método.

{ Qual é a diferença entre recursão direta e indireta?

Uma função **fun** é considerada recursiva direta quando chama a mesma função e recursiva indireta se chama outra, como **fun_new** - neste caso, **fun_new** chama **fun** direta ou indiretamente.

Recursão direta	Recursão indireta

void directRecFun(){
 	
// Algum código....
 	
directRecFun();
 	
// Algum código...
 	
}
	void indiretoRecFun1(){
	
	// Algum código...
	
	indiretoRecFun2();
	
	// Algum código...
	}
	void indiretoRecFun2(){
	
	// Algum código...
	
	indiretoRecFun1();
	
	// Algum código...
	}

{ Como alocar memória para diferentes chamadas de função

Em funções de **main()**, a memória é alocada na pilha. Uma função recursiva chama a si mesma, ou seja, a memória para a função chamada é sobreposta à memória alocada para a função de chamada e uma cópia diferente das variáveis locais é criada para cada chamada de função. Quando o caso-base é atingido, a função retorna seu valor para aquela pela qual foi chamada, a memória é desalocada e o processo continua.

Existem vantagens e desvantagens quanto ao uso de funções recursivas:

Vantagens	Desvantagens
• Maneira limpa e simples de escrever código	• Maiores requisitos de espaço que o programa iterativo, já que todas as funções permanecem na pilha até que o caso–base seja alcançado • Maiores requisitos de tempo devido às chamadas de função e sobrecarga de retorno

{ Programação dinâmica

A técnica de programação dinâmica (ou *dynamic programming*, em inglês) ajuda a resolver de forma eficiente uma classe de problemas com subproblemas sobrepostos e propriedade de subestrutura ótima. Esses problemas incluem o cálculo repetido do valor dos subproblemas para encontrar uma solução ideal.

Como funciona ?

A programação dinâmica armazena o resultado dos subproblemas de forma que, quando soluções forem necessárias, estes estejam disponíveis, não sendo necessário recalculá-los. Essa técnica de armazenar valor dos subproblemas é chamada de “memorização”. Um problema computacional pode ser resolvido por programação dinâmica caso possua duas características: subproblemas sobrepostos ou subestrutura ótima.

Subproblemas sobrepostos

A programação dinâmica combina soluções para subproblemas e é usada majoritariamente quando as soluções destes são repetidamente necessárias. Estas soluções computadas são armazenadas em uma tabela para não precisarem ser recalculadas. A programação dinâmica não é útil quando não há subproblemas comuns (sobrepostos), já que não faz sentido armazenar soluções se elas não forem utilizadas novamente.

Existem duas maneiras de armazenar valores para que estes sejam reutilizados: memorização (*top down*) e tabulação (*bottom up*).

Memorização (top down)

O programa memorizado para um problema é semelhante à versão recursiva exceto por uma pequena modificação, que examina uma tabela de consulta antes de calcular soluções. Ao iniciarmos um *array* de pesquisa com todos os valores iniciais como NIL, sempre que precisarmos de solução para um subproblema, precisamos primeiro examinar a tabela de pesquisa. Se o valor pré-calculado estiver lá, retornamos esse valor; caso contrário, calculamos o valor e incluímos o resultado na tabela de pesquisa para ser reutilizado posteriormente.

Tabulação (bottom up)

O programa tabulado para um determinado problema constrói uma tabela de baixo para cima retornando a última entrada dela. tabulados e memorizados armazenam as soluções dos subproblemas, mas de formas diferentes: na versão memorizada, a tabela é preenchida sob demanda; na tabulada, a partir da primeira entrada, as demais são preenchidas uma a uma. Ao contrário da versão tabulada, as entradas da tabela de pesquisa não são necessariamente preenchidas na versão memorizada.

Subestrutura ótima

Um problema possui propriedade de subestrutura ótima se a solução for obtida por meio de soluções ótimas de seus subproblemas.

Passos para resolver um algoritmo de programação dinâmica

- Identifique se é um problema de programação dinâmica;
- Decida uma expressão de estado com menos parâmetros;
- Formule relação de estado;
- Faça tabulação (ou adicione memorização).

Referências

https://www.geeksforgeeks.org/recursion-in-java/

https://www.programiz.com/java-programming/recursion

https://www.geeksforgeeks.org/dynamic-programming/#concepts

Professora: Jéssica Osko | Monitora: Renata Pereira

Google olabi

<RECURSÃO& PROGRAMAÇÃODINÂMICA>

/exercícios/

{ Exercício coletivo I

Escreva um algoritmo de fatorial de um número usando recursão.

{ Exercício coletivo II:

Escreva um programa que detalhe as diferentes formas de cobrir uma distância. Dada a distância **dist**, conte o total de formas de percorrer a distância em 1, 2 e 3 passos.

{ Extras:

I – Escreva um programa que minimize o número de caixas colocando uma menor dentro de uma maior. Dado um **array size[]** de tamanhos de caixa, a tarefa é encontrar o número de caixas restantes após colocar a caixa menor em uma maior.

Nota: Apenas uma caixa pequena pode caber dentro de uma maior.

II – Escreva um programa para a Tower de Hanoi:

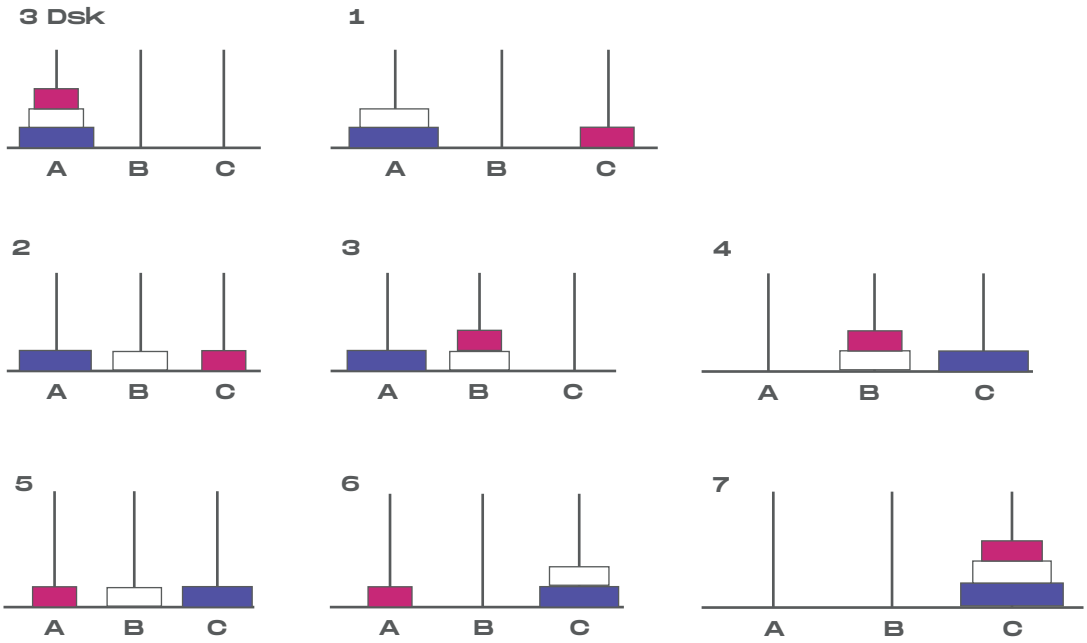
A Torre de Hanói é um quebra-cabeça matemático com três hastes e **n** discos. O objetivo é mover toda a pilha para outra haste obedecendo as seguintes regras:

a) Cada movimento deve consistir em pegar o disco superior de uma das pilhas e colocá-lo em cima de outra;
b) Um disco só pode ser movido se for o mais alto da pilha;
c) Apenas um disco movido por vez;
d) Nenhum disco pode ser colocado em cima de um disco menor.

e) Abordagem:

Considere a haste 1 = 'A'; haste 2 = 'B'; e haste 3 = 'C'.

- Mova os discos 'n-1' de 'A' para 'B' usando C;
- Mova o último disco de 'A' para 'C';
- Mova os discos 'n-1' de 'B' para 'C' usando A.



<Referência para os exercícios>

https://www.programiz.com/java-programming/recursion

https://www.geeksforgeeks.org/