Trabalho 2: Solver de Torres de Hanói com Indução/Recursão

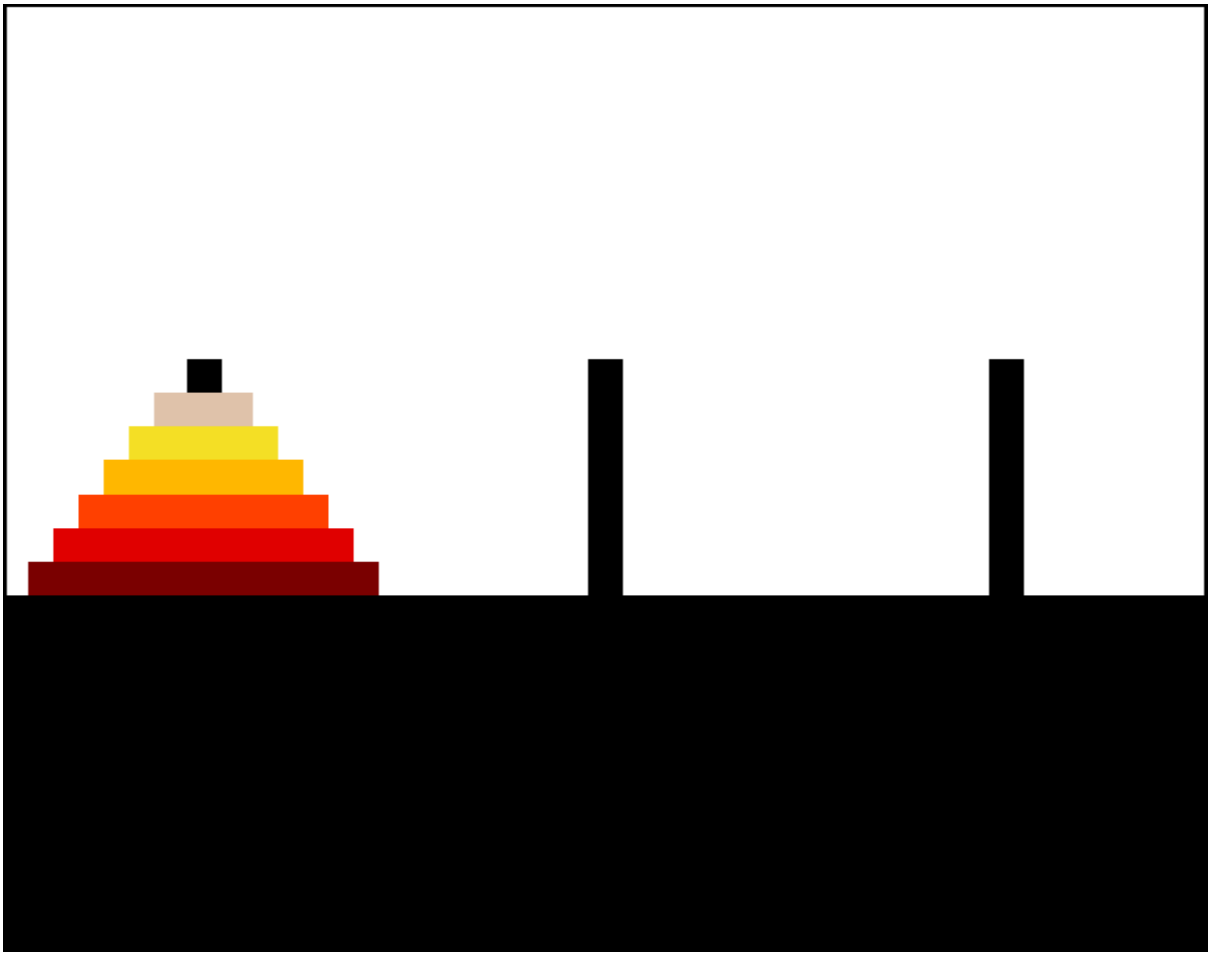
Nomes: Henrique Fugie de Macedo e Luiz Otávio Silva Cunha
Matrículas: 0056151 e 0056153

Torre de Hanói

Torre de Hanói é um quebra-cabeça criado por Édouard Lucas em 1883, o nome é baseado em uma torre na cidade de Hanói no Vietnã. O quebra-cabeça tem uma premissa simples, para jogar o quebra-cabeça é necessário alguns canos e discos de tamanhos distintos, sendo que o tamanho dos discos vai aumentando.

O objetivo da Torre de Hanói consiste em realocar todos os discos de um cano para outro, porém tem certas regras.

1. Só se pode mover um disco de cada vez, ou seja, cada movimento equivale a mudança de um só disco.
2. Disco de tamanhos menores não podem ficar em baixo de discos de tamanho maiores, ou seja semelhante a uma pirâmide, base larga e topo fino.



Como se pode analisar na imagem acima, a “torre” já vem construída e basta apenas transferir todos os discos para um cano posterior, porém deve-se obedecer as regras, o quebra-cabeça termina quando toda a torre foi transportada, porém o maior desafio é fazer essa transferência utilizando o número mínimo de passos.

Estratégia das Torres de Hanói

A estratégia das torres de hanói consiste em setar um dos canos com auxiliar e ir passando os discos menores para ele, assim colocando os discos maiores no cano de destino, como é mostrado no gif acima, o cano do meio foi utilizado como o auxiliar, percebe que para todas as vezes quando se quer mover um disco maior para o cano mais a direita, o do meio recebe os menores, assim garantindo que o mais a direita esteja livre para receber o disco maior.

Indução

A indução é uma técnica fundamental na resolução de problemas computacionais, baseada na ideia de decompor um problema complexo em instâncias mais simples e resolvê-las de maneira incremental. A estratégia de indução requer a identificação de um caso base, que serve como ponto de partida para a resolução de problemas mais complexos.

Identificação do Caso Base:
O primeiro passo na aplicação da indução é encontrar um caso base, uma instância do problema que seja trivial de resolver. Essa instância serve como fundamento inicial para a indução, proporcionando uma solução direta e evidente. A escolha adequada do caso base é crucial, pois ele deve ser simples o suficiente para ser resolvido sem grande esforço e, ao mesmo tempo, representar uma instância básica do problema.

Aplicação da Indução:
Uma vez identificado o caso base, o próximo passo é mostrar como resolver instâncias mais complexas do problema, baseando-se na solução do caso base. Isso é feito por meio do processo indutivo, que consiste em assumir que a solução é válida para um tamanho menor do problema e, em seguida, mostrar como essa solução pode ser estendida para um tamanho maior.

Casos Menores para Casos Maiores:
A transição da solução do caso base para casos mais complexos ocorre através de passos indutivos. O programador precisa demonstrar que se a solução funciona para instâncias menores do problema, então ela também funcionará para instâncias maiores. Isso geralmente é feito através de argumentos lógicos e passos de raciocínio que mostram como a solução pode ser ampliada de uma instância para outra.

Explicação do código e modelagem utilizada

Explicando um pouco sobre o código agora, o código utiliza recursão com indução, uma indução recursiva com uma divisão e conquista por trás de tudo, porém está dentro do que foi pedido no trabalho que é trabalhar em cima da indução. A base do código tem uma estrutura similar com a do código do sudoku, onde ele possui a parte do Menu do código e a parte do solver, que faz a solução do problema, com a adição de uma simples função para mostrar como estão as pilhas, onde estão cada disco. Para armazenar a posição de cada disco, foram utilizadas listas, onde ele armazena em ordem decrescente, do número que o usuário forneceu até o número 1, e todas elas estão dentro de um dicionário que armazena os três pilares da Torre de Hanói, e em cada uma tem a sua lista de discos, foram utilizados listas ao invés de pilhas pela facilidade de manipular os valores dentro das listas, mas manipulando elas como se fossem uma pilha, com append que adiciona no final da pilha e o pop que remove o último elemento da lista, ao invés de ter que utilizar comando diferentes, ou até mesmo outras bibliotecas que resultam em uma dor de cabeça maior.

Outros pontos que foram escolhidos para facilitar a execução do código, como foi solicitado no enunciado do trabalho, ele mostra cada passo da solução, mostrando cada movimento de um disco e sempre mostrando os estados de todos os discos a cada execução, a ideia de “divisão e conquista” também interligado a indução faz com que resolvendo o caso base, de 1 disco, e resolvendo o caso para 2 discos, ele acaba resolvendo os casos para n discos, onde chegou a ser testado a solução com 200 discos e conseguiu encontrar a solução, mesmo demorando um tempo imenso.

Explicando um pouco do código em si agora, mostrando uma saída de exemplo, com 3 discos e cada pino nomeado de acordo com a imagem, vamos explicar como ocorre a recursão no código:

```
Digite o número de discos: 3
Digite o nome do pino de origem: a
Digite o nome do pino auxiliar: b
Digite o nome do pino de destino: c
Estado dos pinos:
a: [3, 2, 1]
b: []
c: []

Resolvendo torre de Hanoi para 3 discos de a para c usando b como auxiliar.
Resolvendo torre de Hanoi para 2 discos de a para b usando c como auxiliar.
Resolvendo torre de Hanoi para 1 discos de a para c usando b como auxiliar.
mover o disco 1 de a para c
Estado dos pinos:
a: [3, 2]
b: []
c: [1]

Resolvendo1 torre de Hanoi para 2 discos de a para b usando c como auxiliar.
mover o disco 2 de a para b
Estado dos pinos:
a: [3]
b: [2]
c: [1]
```

As primeiras linhas de código mostram as entradas fornecidas pelo usuário, junto com a Torre de Hanói que foi fornecida, em seguida, ele chama a função resolver_torre_de_hanoi e começa retornando Resolvendo torre de Hanoi para 3 discos (apenas na versão de debug) onde o objetivo do código é mover todos os discos para a torre de destino C com o auxiliar B, em seguida manda um retorno parecido, porém para atingir a saída anterior, ele deve mover os outros 2 discos, que são chamados de forma recursiva para a Torre B, utilizando a C como auxiliar que é feito com uma nova chamada recursiva, por isso duas chamadas recursivas. Em seguida temos a solução para 1 disco, que neste caso já entra no caso base, os destinos são alternados a fim de não colocar um disco com valor maior em cima de um disco com valor menor, e a recursão já cuida também desse ponto, de forma que o disco maior só seja movido quando o movimento do menor ocorrer. Em seguida, a chamada de 2 discos de b para c retorna, após ter sido feito o movimento do número 1 para a torre C, chamada pelo retorno Resolvendo1, onde ele teve que voltar para solucionar uma recursão “feita” anteriormente para acabar de resolver, como se estivesse desempilhando uma chamada recursiva anterior para resolver os subproblemas menores, e ele vai repetindo isso até solucionar o problema, podendo resolver até n discos do problema.