1 Recursão - Aula 1

1. **Fatorial recursivo**: Implemente uma função recursiva para calcular o fatorial de um número.

2. **Soma recursiva dos elementos de uma lista**: Implemente uma função recursiva para calcular a soma de todos os elementos de uma lista.

3. **Fibonacci recursivo**: Implemente uma função recursiva e outra iterativa para calcular o k-ésimo elemento da sequência de Fibonacci. Há alguma diferença significante no tempo de processamento das duas? Por quê?

2 Recursão - Aula 2

4. **Exponenciação recursiva**: Implemente uma função recursiva para calcular o valor de x^n.

5. **Maior elemento de uma lista recursivo**: Implemente uma função recursiva para encontrar o maior elemento de uma lista.

6. **Máximo Divisor Comum recursivo**: Implemente uma função recursiva para calcular o Máximo Divisor Comum entre dois números.

7. **Inversão**: Implemente uma função recursiva para imprimir ao contrário todos os algarismos de um número inteiro.

3 Recursão - Aula 3

8. **Torre de Hanoi**: Num grande templo na Índia, há uma placa onde estão fixados três pinos de diamante. Diz a lenda que num deles, no momento da criação, o deus Brahma colocou 64 discos de ouro puro, o maior deles na base e os restantes na ordem decrescente de tamanho até o topo. Os monges deveriam se revezar, transferindo os discos de um pino para outro, obedecendo às regras descritas acima. Quando os 64 discos fossem transferidos do pino em que Deus os colocou para qualquer um dos outros dois, o templo viraria pó e o mundo desapareceria.

Figura 1: Torre de Hanoi.

Considerando que as três torres se chamam, respectivamente, A, B e C, crie uma função recursiva para imprimir a menor sequência de passos necessária para mover n discos de A para B, podendo usar o auxílio de C. Regras:

(a) Só é possível mover um disco de cada vez.

(b) Um disco nunca pode ficar acima de outro disco menor que ele.

Veja uma simulação online do jogo em `https://www.matematica.pt/fun/hanoi.php` .

4 Recursão - Aula 4

1. **Permutações - Impressão**: Implemente e teste uma função que imprima todas as permutações de uma lista l.

2. **Permutações**: Implemente e teste uma função que retorne uma lista com todas as permutações entre os elementos de uma lista l.

5 Recursão - Extra

3. **TSP**: No Problema do Caixeiro Viajante (do inglês, *Travelling Salesman Problem*, ou apenas *TSP*), um comerciante precisa visitar vários locais, partindo de um local inicial qualquer, passando por todos os locais exatamente uma vez e voltando ao local inicial no fim do percurso. Implemente uma função que calcule o caminho de menor custo total que o caixeiro pode fazer para uma lista de locais que é recebida como parâmetro. Considere que cada local é uma tupla contendo o nome de uma

cidade e sua posição no plano cartesiano, e que o custo entre dois locais é a distância Euclidiana entre eles.

6 Laboratório 11

4. **Vizinho Mais Próximo**: Um dos algoritmos mais famosos para encontrar uma solução para o TSP num tempo satisfatório é o "Vizinho Mais Próximo" (Nearest Neighbour, ou **NN**). No algoritmo NN, o *tour* se inicia com um local qualquer. Enquanto não se insere todos os locais, deve ser escolhido o destino mais próximo do último local inserido, dentre todos os demais locais que ainda não estão no *tour* – daí o nome vizinho mais próximo. Esse destino mais próximo é então inserido no *tour*. Quando não houver mais locais a serem inseridos, o algoritmo acaba.

   Por exemplo, considere os quatro locais a serem visitados, mostrados no plano abaixo:

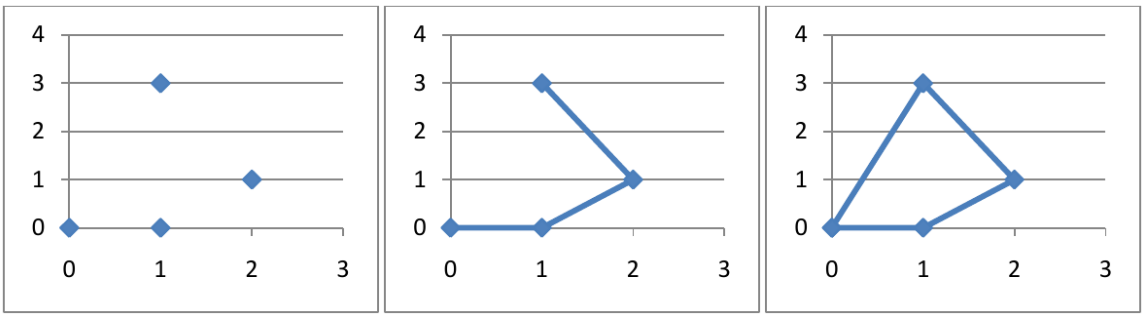


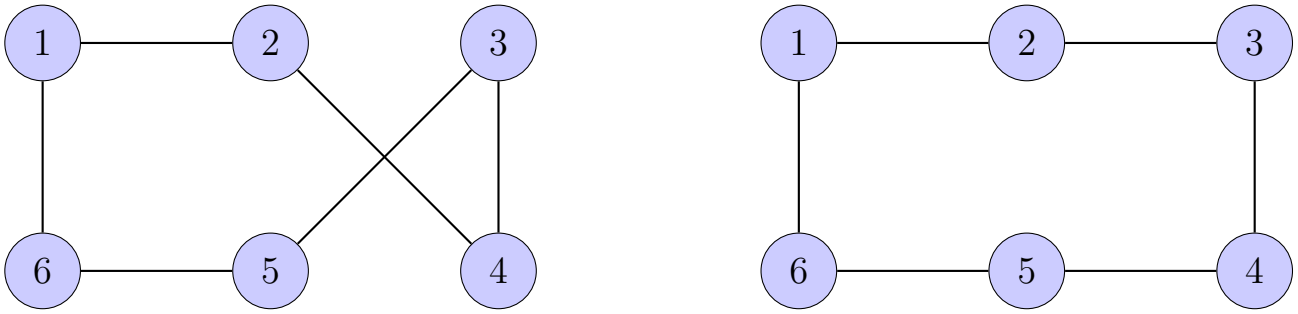
Figura 2: Passos do algoritmo Vizinho Mais Próximo.

   A primeira parte da figura mostra a representação dos pontos $(0.0, 0.0)$, $(1.0, 3.0)$, $(2.0, 1.0)$ e $(1.0, 0.0)$, que devem ser visitados pelo caixeiro. A segunda parte mostra a ordem em que ele visitará esses locais. A terceira mostra o caminho total, incluindo a volta ao local inicial no fim do percurso.

   O primeiro passo do NN seria escolher o local inicial. No caso do exemplo, o local de partida é o ponto $(0.0, 0.0)$. Em seguida, é inserido no *tour* o local mais perto do último que foi inserido. No exemplo, o local mais perto de $(0.0, 0.0)$ está no ponto $(1.0, 0.0)$. O próximo a ser inserido está no ponto $(2.0, 1.0)$, e assim sucessivamente.

   Calcule o caminho gerado pelo algoritmo *NN* para uma lista de cidades que é recebida como parâmetro, considerando que cada local é representado por uma tupla contendo o nome de uma cidade e sua posição no plano cartesiano, que o custo entre duas cidades é a distância Euclidiana entre elas, e que a cidade de partida é a primeira da lista recebida como parâmetro.

5. **2-OPT**: 2-OPT é um algoritmo de busca local para melhorar que tenta aprimorar uma solução prévia do TSP. A ideia principal é pegar uma rota que cruze ela mesma e reordená-la de forma que isso não ocorra. Exemplo:



   Uma busca 2-OPT completa vai comparar todas as combinações possíveis de trocas entre duas arestas não-adjacentes. A melhor troca é efetuada, e a busca é então refeita até que não haja trocas que melhorem o *tour*.

   Dado um caminho (por exemplo, gerado pelo Vizinho mais Próximo), calcule a rota gerada pelo 2-OPT.

6. **Exercícios extras de recursão**:

   (a) Crie uma função recursiva que receba um número inteiro positivo N e calcule o somatório dos números de 1 a N.

   (b) Crie uma função recursiva que receba uma lista e inverta a ordem dos elementos dessa lista.

   (c) Crie uma função recursiva que receba k e n, e determine quantas vezes o dígito K ocorre em um número natural N. Por exemplo, o dígito 2 ocorre 3 vezes em 762021192.