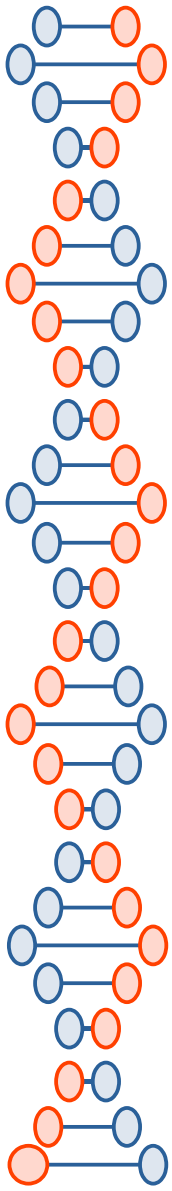


Estrutura de dados 1 – Uniube

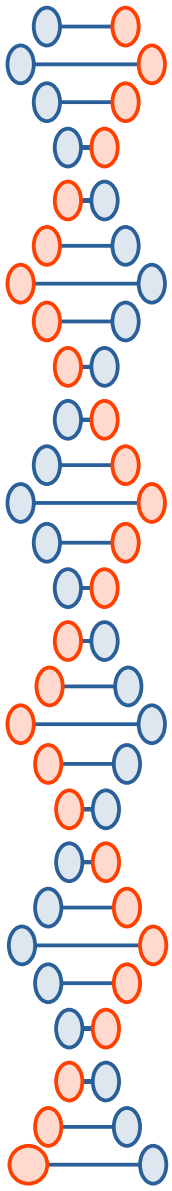
Prof. Dr. Marcos Lopes

(34) 9 9878 0925

malopes21@gmail.com

https://sites.google.com/view/malopes21/

AULA 01



Ementa:

Estudo dos conceitos das estruturas estáticas e dinâmicas e suas aplicações para o armazenamento de dados.

Manipulação de estruturas dinâmicas lineares e não lineares.

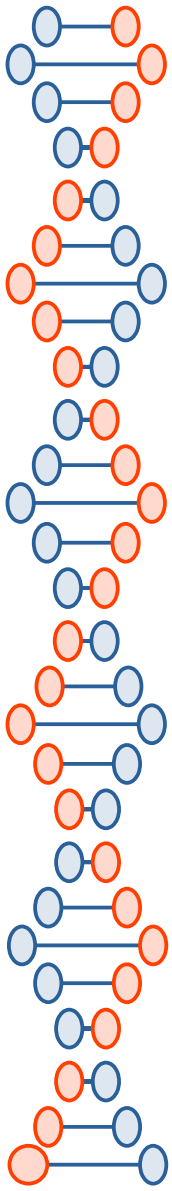
Bibliografia básica:

ASCENCIO, Ana Fernanda Gomes; ARAUJO, Graziela Santos de. Estruturas de dados: algoritmos, análise da complexidade e implementações em Java e C/C++. https://uniube.bv3.digitalpages.com.br/users/publications/9788576058816/pages/_1.

CORMEN, Thomas H. Algoritmos:teoria e prática. Rio de Janeiro: Elsevier, 2012.

DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo (SP): Thomson, 2002,

PREISS, Bruno R. Estrutura de dados e algoritmos: padrões de projetos orientados a objetos com Java. Rio de Janeiro (RJ): Campus, 2001. 566 p.



Estruturas de dados:

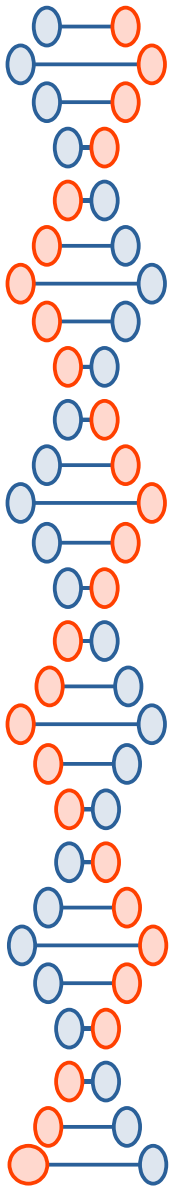
As estruturas de dados são o coração de qualquer programa sofisticado.

Selecionar a estrutura de dados correta pode fazer uma grande diferença na complexidade da implementação resultante.

Escolha a representação de dados adequada, e sua tarefa será fácil de programar.

Escolha a representação errada, e você pode gastar enormes quantidades de tempo e de código encobrindo sua decisão inicial ruim.

Vamos analisar as estruturas de dados fundamentais que todo programador deve estar familiarizado.



Estruturas de dados elementares:

Aqui apresentamos as operações abstratas das estruturas de dados mais importantes, como:

arranjos, listas, pilhas, filas, mapas associativo (dicionários), conjuntos, árvores, grafos (outros ???).

Descreveremos uma forma simples de implementar estas estruturas a partir do zero.

Esteja ciente, no entanto, que as linguagens orientadas a objetos modernos, vêm com bibliotecas padrão de estruturas de dados fundamentais.

Mas vale a pena o tempo de cada programador de se familiarizar com as suas bibliotecas em vez de reinventar a roda repetidamente.

Uma vez que você fizer isso, você pode ler esta seção com um olho para o que cada estrutura de dados é adequada, em vez de os detalhes de como deveria ser implementada.

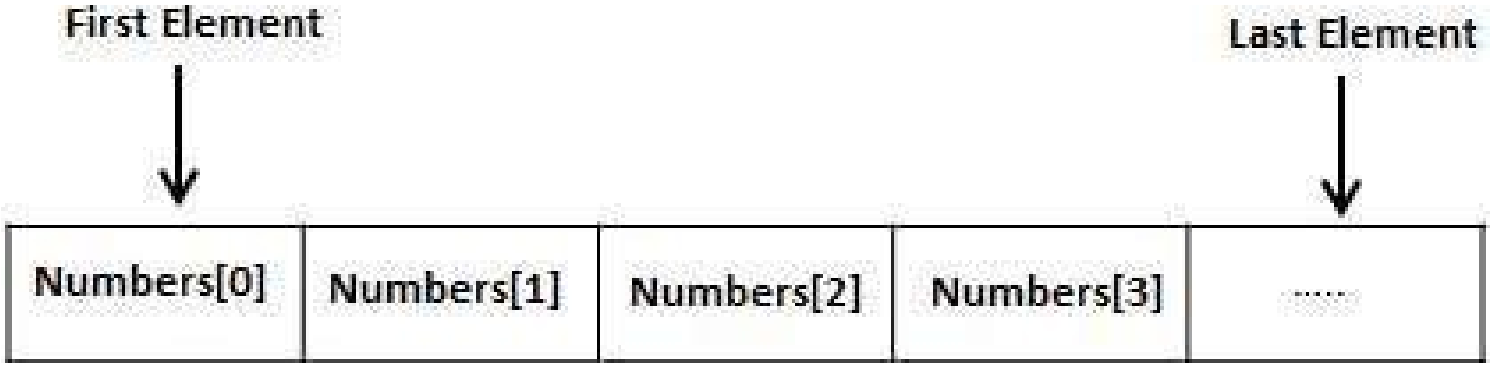
Exemplos:

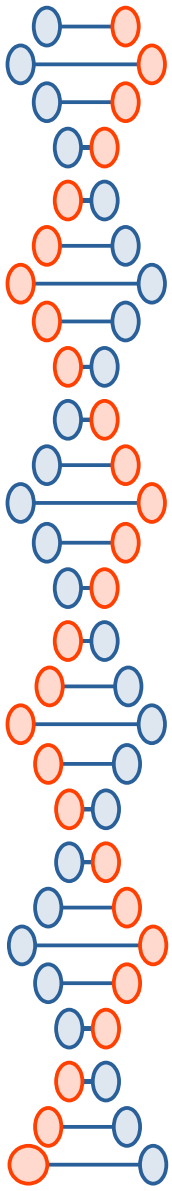
Arrays (Arranjos):

Um arranjo é uma estrutura de dados (alguns autores não consideram como), extremamente elementar

que representa uma organização de dados de mesmo tipo de forma contínua, ou seja um arranjo.

Uma tarefa comum em programação é a manutenção de um conjunto numerado de objetos relacionados (do mesmo tipo).





Em C++, teríamos algo como:

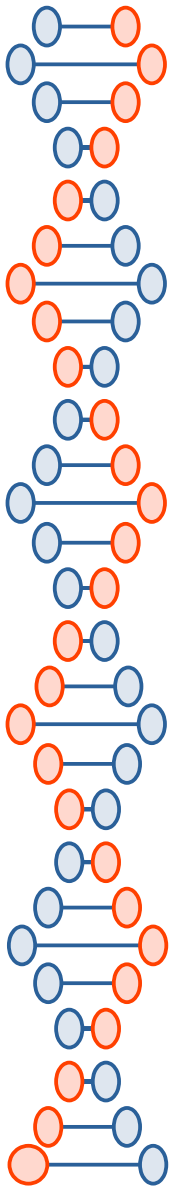
```
#include <stdio.h>

int main ()
{
  int n[ 10 ];
  int i,j;

  for ( i = 0; i < 10; i++ )
  {
    n[ i ] = i + 100;
  }

  for (j = 0; j < 10; j++ )
  {
    printf("Element[%d] = %d\n", j, n[j] );
  }

  return 0;
}
```



Em Java, talvez:

```
public class Classe01 {

    public static void main(String[] args) {

        int[] numeros;
        numeros = new int[10];
        numeros[0] = 940;
        numeros[1] = 880;
        numeros[2] = 830;
        numeros[3] = 790;

        for(int numero: numeros) {
            System.out.print(numero + " ");
        }

        for(int i=0; i<numeros.length; i++) {
            System.out.print(numeros[i] + " ");
        }
    }
}
```

Listas (Lists):

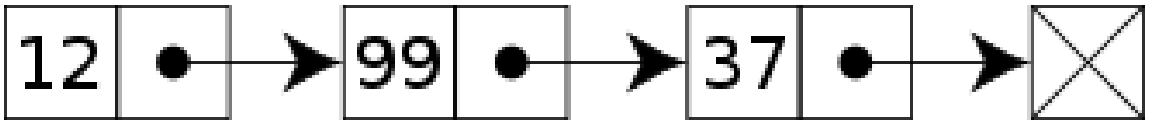
Podemos definir a estrutura de dados lista como sendo arranjos dinamicamente reindexados e dinamicamente realocados.

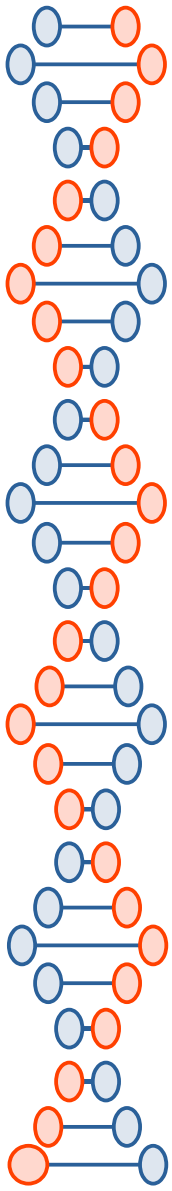
Os elementos de uma lista são acessados por um índice e de modo geral sua inserção é realizada no final (como uma lista de itens mesmo).

Algumas operações como:

add(x): adiciona um elemento no final da lista
remove(Obj): remove um objeto da lista
indexOf(Obj): retorna o indice de um objeto na lista

Pode-se optar pela implementação com elementos encadeados, como:





Pilhas (Stacks):

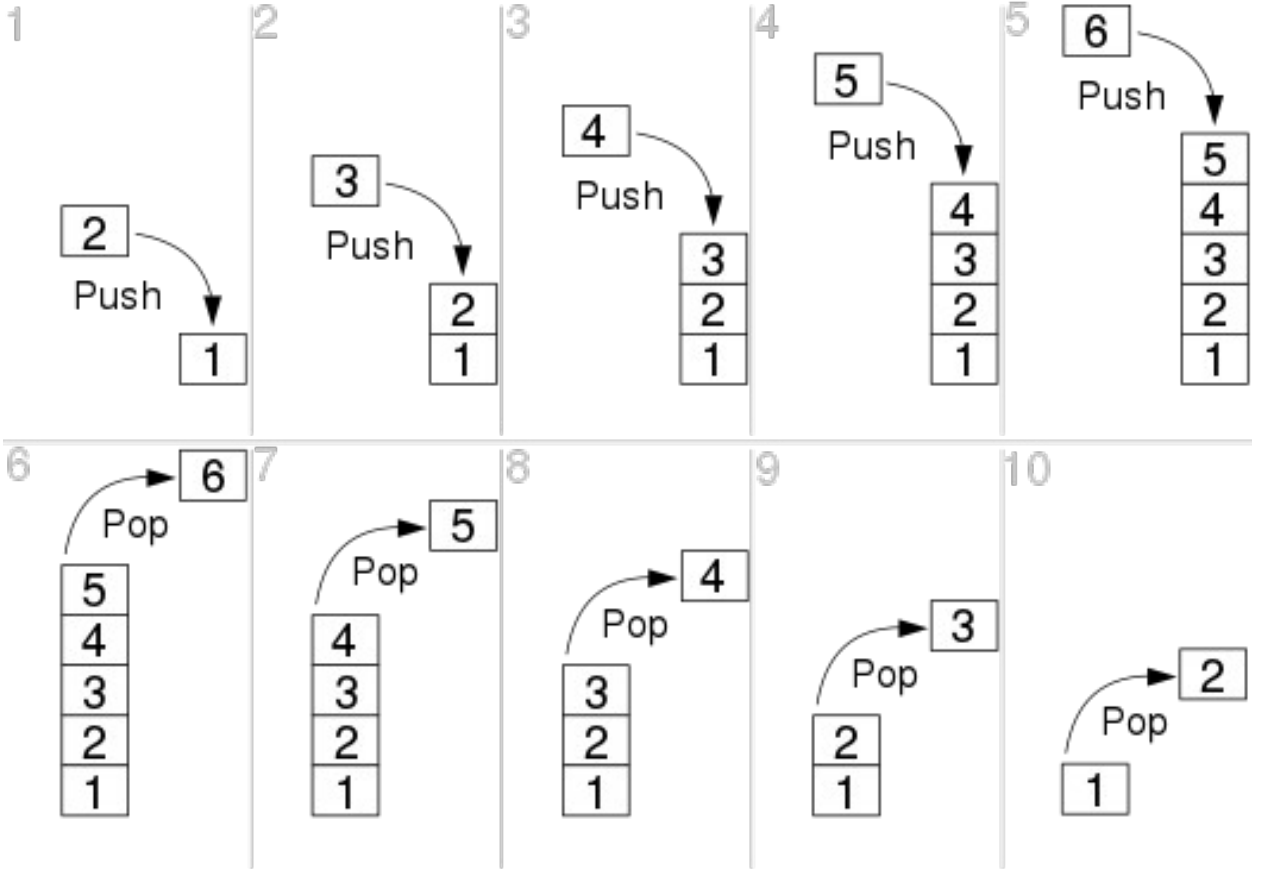
Pilhas e filas são recipientes onde os itens são recuperados de acordo com a ordem de
inserção, independente do conteúdo.

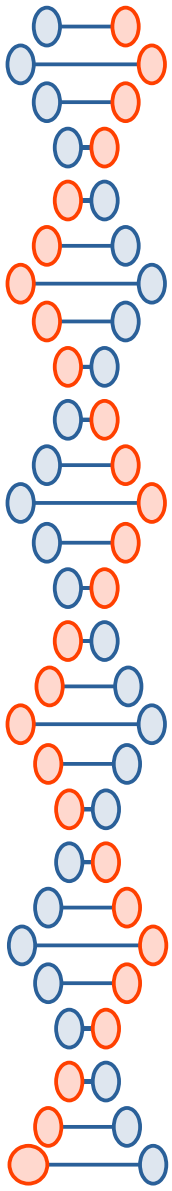
Pilhas mantem o padrão LIFO (last in first out), último a entrar, primeiro a sair.
Em resumo as operações sobre uma pilha incluem :

• push (x, s) - Insert x item no topo da pilha s.
• pop (s) - retorna (e remove) o item do topo da pilha s.
• inicializar (s) - criar uma pilha vazia.
• vazio (s) - Testar se a pilha pode aceitar mais pushs ou pops, respectivamente.

Note-se que não há nenhuma operação de busca de elemento definido em pilhas.

Pilhas (Stacks):



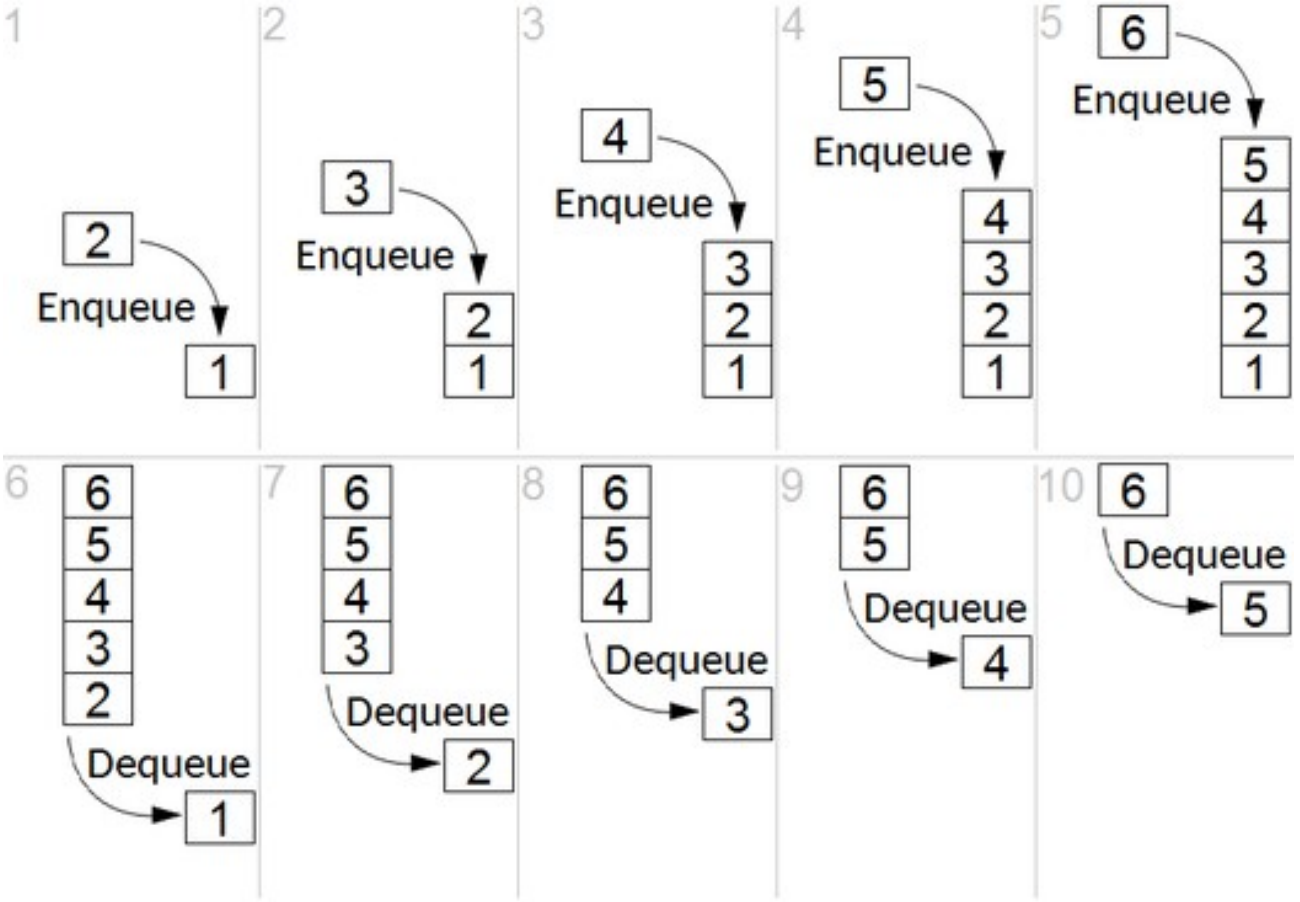


Filas (Queues):

Uma estrutura de dados do tipo fila representa exatamente o conceito de uma fila como estamos habituados e ver: fila no banco, fila no supermercado, etc.

É uma estrutura de dados do tipo FIFO (first in first out): primeiro a entrar é o primeiro a sair.

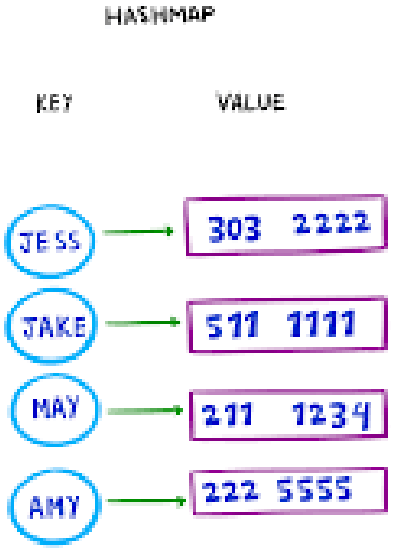
Filas (Queues):



Mapas associativos (dicionários):

Um mapa associativo, ou um dicionário, representa uma estrutura de dados que armazenas pares que se associam no modelo (chave -> valor), ou seja, cada par é armazenado num conjunto e são recuperados pela chave.

As chaves de modo geral são codificadas com modelos de tabelas hash para permitir uma leitura extremamente eficiente.

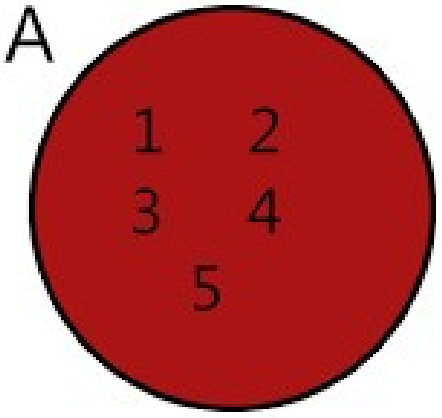


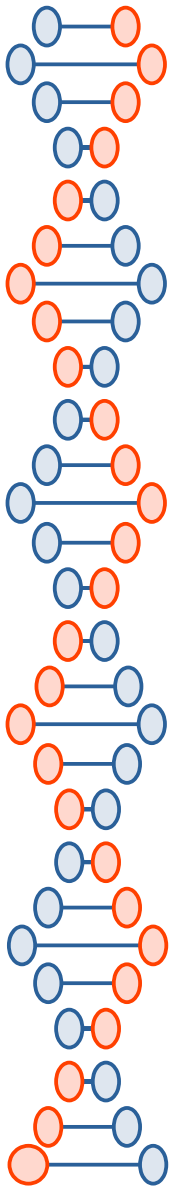
Conjuntos:

Os conjuntos (sets) são estruturas de dados que representam o conceito de conjuntos da matemática, ou seja, não existem elementos duplicados e os elementos num conjunto não são indexados.

De modo geral são não ordenados até mesmo pela ordem de entrada, mas podem ter esse recurso caso desejado.

Exemplo:





Árvores:

Podemos definir uma estrutura de dados do tipo árvore como um grafo acíclico.

Um grafo é uma estrutura simples por definição que representa a junção de elementos (nós) através de arestas (ligações).

Essas ligações não são limitadas em número nem mesmo em ordem.

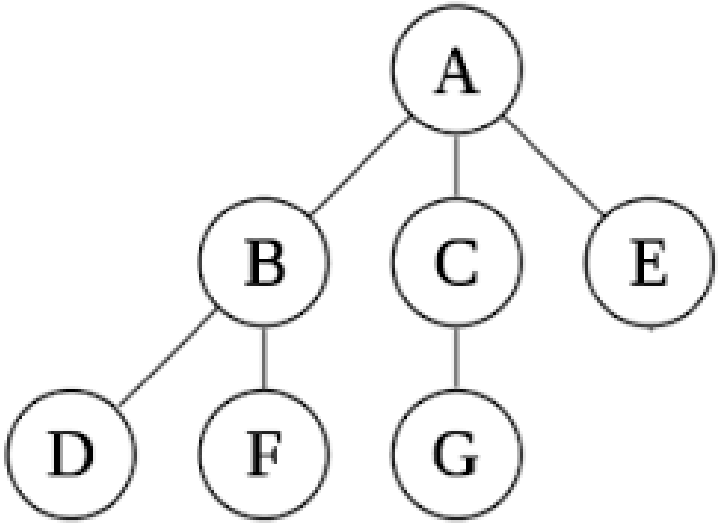
De modo geral é uma estrutura de dados adequada na modelagem de dados de uma rede social por exemplo.

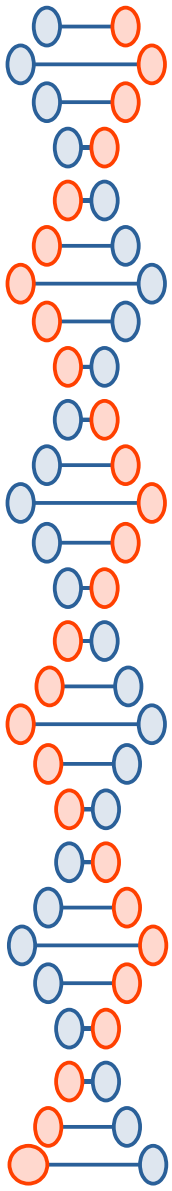
Já as árvores são grafos com 2 limitações:
- sempre existe um nó (elemento) raiz que não tem ligação chegando
- não é permitido ciclo nas ligações dos elementos

ou seja: um grafo acíclico!

Árvores:

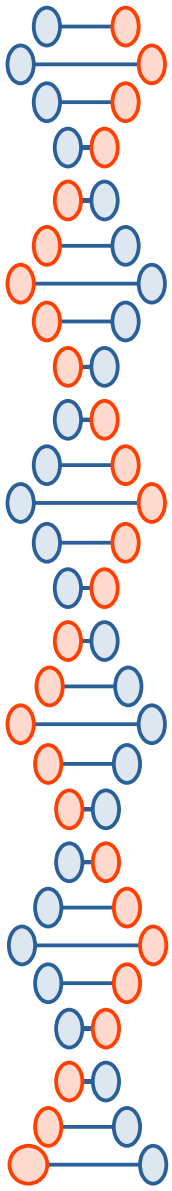
Exemplo:





Exercícios para aquecimento:

Exercício 1) Construir um programa que lê dois arranjos de 5 posições cada (cada um representando um vetor) e calcula do produto escalar entre esses aranjos. Criar e usar no programa, uma função/método, int calculaProdEscalar(int[], int[]), que recebe dois vetores/arranjos de inteiros, calcula o produto escalar e retorna com o valor.

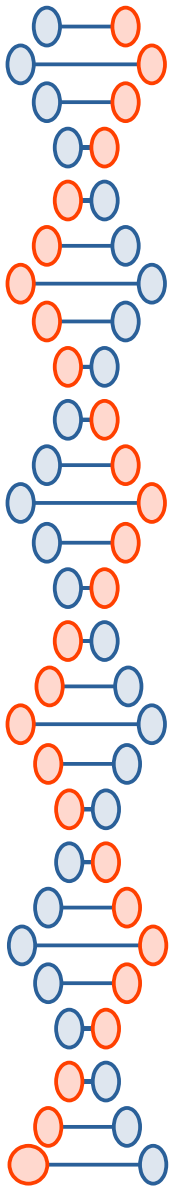


Exercício 2) Beecrowd | 1214 (Acima da Média)

Sabe-se que 90% dos calouros tem sempre a espectativa de serem acima da media no inicio de suas graduacoes. Voce deve checar a realidade para ver se isso procede.

Entrada A entrada contem muitos casos de teste. A primeira linha da entrada contem um inteiro C, indicando o numero de casos de teste. Seguem C casos de teste ou instancias. Cada caso de teste inicia com um inteiroN, que e o numero de pessoas de uma turma (1 ≤ N ≤ 1000). Seguem N inteiros, separados por espacos, cada um indicando a media final (um inteiro entre 0 e 100) de cada um dos estudantes desta turma.

Saída Para cada caso de teste imprima uma linha dando o percentual de estudantes que estao acima da media da turma, com o valor arredondado e com 3 casas decimais.



Exemplo de Entrada

5

5 50 50 70 80 100

7 100 95 90 80 70 60 50

3 70 90 80

3 70 90 81

9 100 99 98 97 96 95 94 93 91

Exemplo de Saída

40.000%

57.143%

33.333%

66.667%

55.556%

Exercício 3) Construir um programa com um arranjo de 10 posiões de inteiros. Preencher o array/arranjo com 10 recordes quaisquer diretamente no código. O programa oferece um menu para o usuário com as opções:

1 – Entrar recorde;

2 – Ver todos os recordes;

3 – Ordenar por score (usando ordenação por seleção)

4 – Sair

Obs1: implementar as funcionalidades do menu;

Obs2: como o arranjo armazena apenas 10 posições, cada recorde entrado "empurra pra fora" um recorde já existente quando o arranjo está completamente cheio.