Git: https://github.com/icei-pucminas/aeds2

Aula 01/08/2022

* Tipos de alocações:
  + Linear àVetor com tamanho definido, alocação limitada na memória.
  + Flexível àVetor com tamanho indefinido, alocação ilimitada na memória.
* Analise de algoritmos – pesquisar n em um array de 1024:
  + Usando o sistema de pesquisa convencional, precisa-se de 1024 MAX
  + Usando o sistema de pesquisa de arvore **balanceada**, precisa-se de log 1024, ou seja, 10 pesquisas no MAX – SEMPRE NA BASE 2
  + Usando o sistema de tabela, precisa-se de 1,2,3 MAX
* **Unidade I: Introdução – Noções de complexidade**
  + Necessidade de contar para calcular preço dos algoritmos.
  + Tipos de caso:
    - Melhor àmenos tempo
    - Pior àmaior tempo
    - Media(ou esperado)àmédia dos tempos de execução para todas entradas possíveis(abordado em grafos/paa)
  + Quando se tem um comando de ||(or) e o primeiro é verdadeiro, o segundo não será testado.
  + Quando se tem um comando de &&(and) e o primeiro é falso, o segundo não será testado.
  + Contagem de operações:
    - Quando tivermos um laço repetindo na contagem de operações, em que o contador começa com 0, repete até <N, e é incrementado de 1 em 1, faremos N repetições.
    - Será o custo da condição mais o número de interações multiplicado pela soma dos custos da condição e da lista a ser repetida(WHILE). Custo: condição() + n x [lista() + condição()]

Onde n é o N de vezes que o laço é repetido.

Proxima aula – Unidade 1b até pg 89.

Estudo Individual 01/08/2022

* Contagem de operações usando IF:

Melhor caso: condição() + mínimo(listaVerdadeiro(), listaFalso())

Pior caso: condição() + máximo(listaVerdadeiro(), listaFalso())

* Contagem de operações usando FOR:

Quando tivermos uma estrutura de repetição em que o contador começa com zero, repete enquanto menor que n e é incrementado em uma unidade, faremos n iterações I = 0

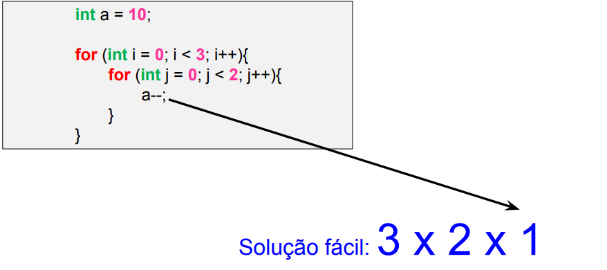
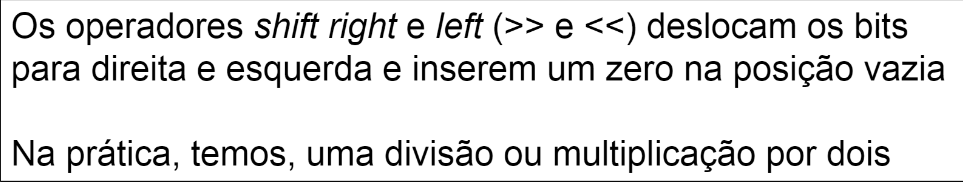
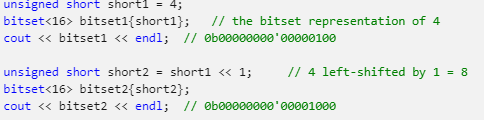
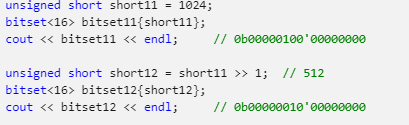
Quando tivermos uma estrutura de repetição em que o contador começa com a, repete enquanto menor que n e é incrementado em uma unidade, faremos (n - a) iterações I = A

* Contagem de operações usando WHILE:

Custo: condição() + n x [lista() + condição()] onde n é o número de vezes que o laço será repetido

* Contagem de operações usando DO:

Custo: n x [lista() + condição()] onde n é o número de vezes que o laço será repetido

* Exemplo de laço dentro de laço:
* Operadores de deslocamento para a esquerda e para a direita (<< e >>)
  + >> desloca bit para direita
  + << desloca bit para esquerda
  + EX:
* Sempre que o tamanho de um problema for, sistematicamente, dividido por dois, temos um custo logarítmico EX:

[n = 4] => 4 2 1 (3 vezes)

[n = 5] => 5 2 1 (3 vezes)

[n = 6] => 6 3 1 (3 vezes)

[n = 7] => 7 3 1 (3 vezes)

[n = 8] => 8 4 2 1 (4 vezes)

[n = 9] => 9 4 2 1 (4 vezes)

[n = 10] => 10 5 2 1 (4 vezes)

[n = 11] => 11 5 2 1 (4 vezes)

[n = 12] => 12 6 3 1 (4 vezes)

[n = 13] => 13 6 3 1 (4 vezes)

[n = 14] => 14 7 3 1 (4 vezes)

[n = 15] => 15 7 3 1 (4 vezes)

[n = 16] => 16 8 4 2 1 (5 vezes)

[n = 17] => 17 8 4 2 1 (5 vezes)

[n = 31] => 31 15 7 3 1 (5 vezes)

[n = 32] => 32 16 8 4 2 1 (6 vezes)

[n = 33] => 33 16 8 4 2 1 (6 vezes)

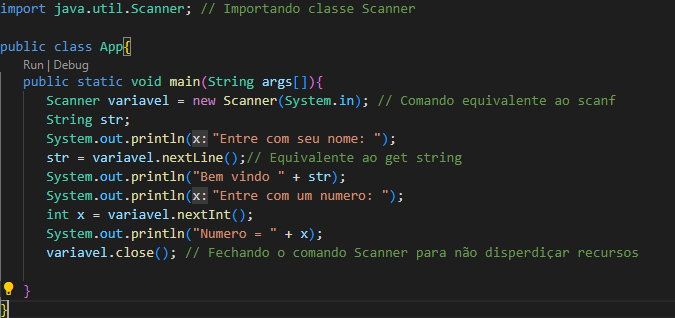
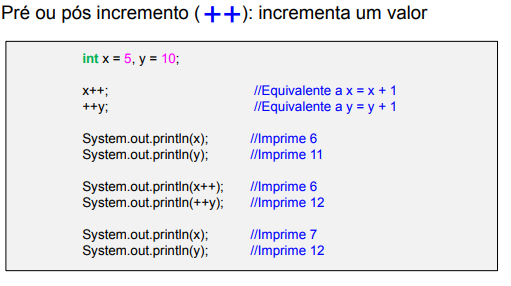
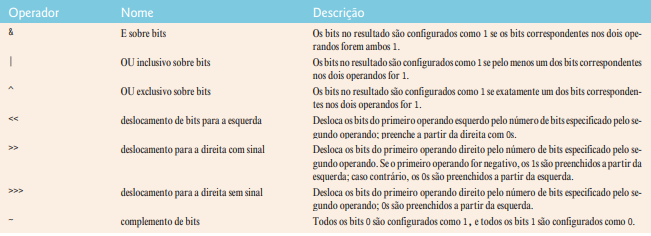
[n = 63] => 63 31 15 7 3 1 (6 vezes)

[n = 64] => 64 32 16 8 4 2 1 (7 vezes)

[n = 65] => 65 32 16 8 4 2 1 (7 vezes)

* Quando i = n e quando n é uma potência de 2, realizamos lg(n) + 1 multiplicações

**Introdução ao JAVA**

* Um programa java é compilado como bytecodes(código objeto para JVM)
* JVM àprocessador virtual que interpreta os bytescodes. à multiplataforma, porém não tão rápido
  + JIT(Just in Time Compiler) = perde multiplataforma e aumenta código
  + JVM em hardware = picoJava,microJava ...
* Linguagem C-like, ou seja , Declaração de variáveis, comentários e sintaxe do if, while,do-while, for, switch-case é igual em C, C++, C# e Java
* Classe principal deve ter o mesmo nome do arquivo.java
* Como compilar e rodar programas java no cdm:
  + Java c NOMEARQUIVO.java à cria um .class
  + Java NOMEARQUIVO
* Como ler dados usando a Classe Scanner:
  + Importar classe : 
  + Pre ou pos incremento:
  + Operações para manipulação de BITS:

Aula dia 03/08/2022

* Unidade I – Introdução
* **Noções de complexidade 1b:**
  + Sempre existirá um custo para cada operação/interação.
  + Teto = / para cima – arredondar para número inteiro.
  + Piso = / para baixo – arredondar para número inteiro.
  + Para realizarmos logaritmos não inteiro devemos sempre pensar na base 2
  + Contagens de estruturas de repetição – DICA: substituir N por números.
    - **FOR:**

1 sit à for(int y = 0 ; y< 5 ; y++) 0 – 5 == 5 operacoes n vezes

2 sit à for(I = 3; i< n; i++) N – 3

Quando tivermos uma estrutura de repetição em que o contador começa com A é repetido enquanto menor que N e incrementado em 1 , teremos N – A repetições.

3 sit à (for dentro do for ) multiplicação das operações de cada laço

4 sit àfor(int i = n; i> 0; i/=2)

Sempre que o escopo do nosso problema for **sistematicamente reduzido pela metade, faremos Ɵ(LGN**).

Podemos ler o operador Ɵ como +/- ou aproximadamente.

Reduz a necessidade de várias operações

Quando i = n e quando n é uma potência de 2, realizamos lg(n) + 1 multiplicações

Divisão será a base do log

* + - **WHILE**:

Custo = condição() + n \* [lista() + condição()]

* + - **DO-WHILE**
    - Custo = n \* [ lista() + condição()]
  + Exercicio em sala:

Quantas repetições são feitas nos laços abaixo:

1. For(i = 0; i< n ; i++) N
2. For(i = 0; i<=n; i++) N + 1
3. For(i = 1; i<n ; i++) N -1
4. For(i = 1; i<=n; i++) N
5. For(i = n ; i>0; i--) N
6. For(i=n; i>=0; i--) N+1
7. X+=(a>b)?a:a+b; à X = (COND) ? V1:V2; Operator alternado 1 ou 2 operações.

* O custo da **pesquisa sequencial é Ɵ de N** , e o custo da **pesquisa binária é Ɵ(LGN**).
  + Pesquisa sequencial = AEDSI , passando por todos os algoritimos do vetor.
  + Pesquisa binaria = AEDSII, Max não explicou ainda.
* O limite inferior(custo mínimo) do **problema de ordenação interna é Ɵ(N \* LGN)****.**Vários algoritmos de ordenação alcançam esse limite e, por isso, são chamados de **ótimos.** Por exemplo, Quicksort , Heapsort e MergeSort.

Aula 08/08/2022

* Para fazer n/2 vezes temos varias opções como: for até n/2 , for e if,...
* Em AEDSII a operação mais **relevante (a que demora mais**) é a comparação entre elementos do array.
* **Ω O Ɵ à** aproximadamente
* Nas notações ignoramos a constante:
  + Um algoritmo que realiza 2,4 e 5 é **Ω O Ɵ(1)**
  + Um algoritmo que realiza 2n,4n e 5n é **Ω O Ɵ(n)**
* Nas notações ignoramos termos com menor crescimento:
  + Um algoritmo que realiza 3n + 2n^2 é **Ω O Ɵ(n^2)**
  + Um algoritmo que realiza 5n+4n^3 é **Ω O Ɵ(n^3)**
* Foto das curvas de crescimentos
* Exercício:

Apresente a notação de complexidade das equações abaixo:

* + - * 1. 5n^2 + 2n + 3 à Ω O Ɵ(n^2)
        2. 99n^20 + 5n^5 + 1111n à Ω O Ɵ(n^20)
        3. 7n^2 + n^4 + 90n^3à Ω O Ɵ(n^4)
        4. 15n^9 + 20n^20 à Ω O Ɵ(n^20)
        5. 70n + 40n^2 + n^3 à Ω O Ɵ(n^3)
        6. Lg n + 2 à Ω O Ɵ(lg n)
        7. 3 lg n + n^2 à Ω O Ɵ(n^2)

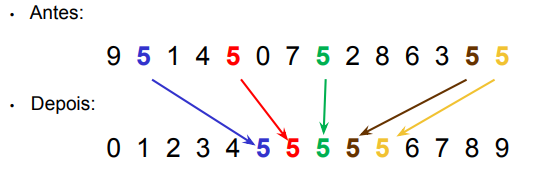
**Algoritmos de ordenação por seleção 1d**

Introdução sobre Ordenação Interna

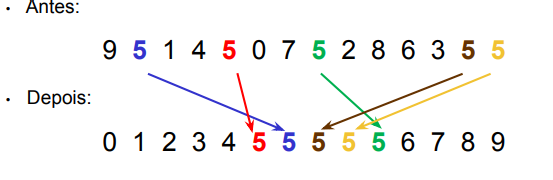
* + Problema de colocar um vetor em ordem.
  + A ordenação é interna quando a lista de elementos cabe na memória principal, caso contrário é externa.
    1. Memoria primaria à RAM
    2. Memoria secundaria à HD/SSD
    3. Chave de pesquisa à atributo usado para ordenar a coleção.
    4. Atributo à algo(propriedade) dos elementos da minha coleção.
    5. Registroàum elemento da coleção

Algoritmos estáveis e não estáveis

* + Um algoritmo é dito **estável** se depois da execução dele, os elementos com a mesma chave mantiverem a ordem relativa entre as chaves repetidas.



* + Um algoritimo é dito **não estável quando:**



Funcionamento básico

* + Procure o menor elemento do array
  + Troque a posição do menor elemento com o primeiro
  + Volte ao primeiro passo e considere o array a partir da próxima posição

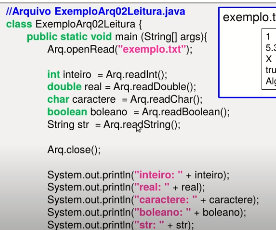
Exercicio próxima aula:

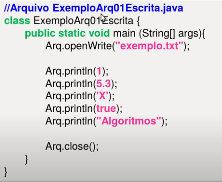
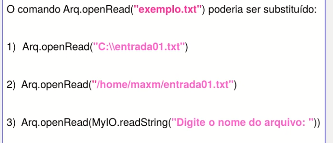
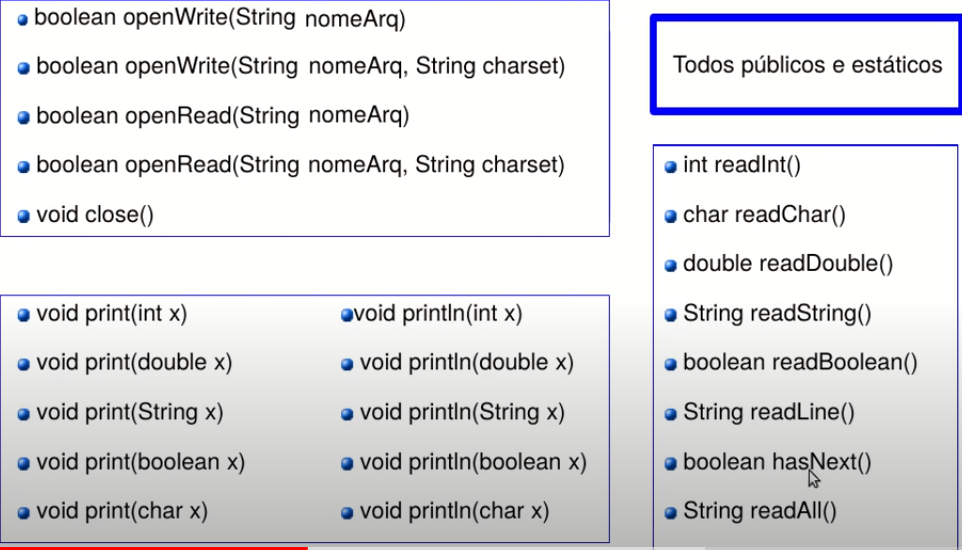
Unidade 1d pg 27 a 35

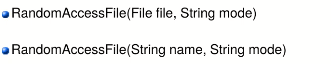
Estudo Individual 08/08/2022

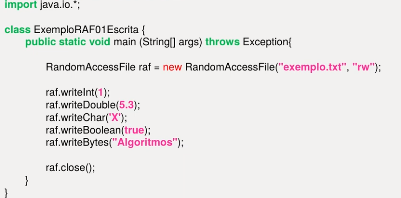
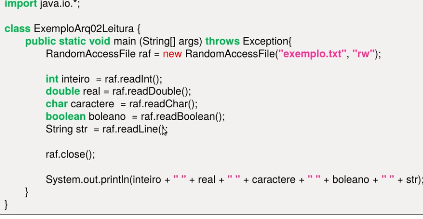
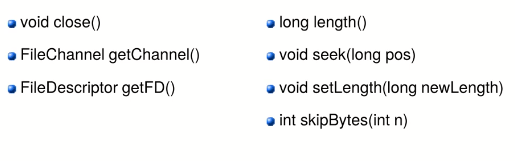
Arquivos em java

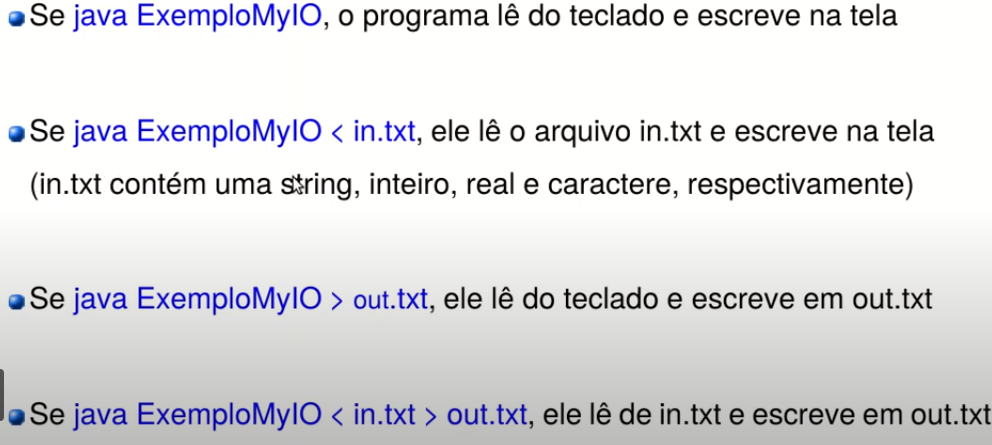
* Classe
  + Arq(modo texto) – acesso sequencial
  + RandomAccessFile(modo binário) – acesso aleatório
* ARQ
  + Utiliza Formatter e Scanner
  + Podem ser lidos com editores de texto e vice-versa



* Exemplos:
* Metodos da classe:
* Classe RandomAccessFile
  + Considera que cada arquivo é um array de bytes indexado por um cursor
  + Após uma operação de leitura/escrita, ele anda no array o número de bytes lidos/ escritos
  + Metodo getFilePoniter à retorna sua posição corrente
  + Metogo void seek(long) à altera esse valor
  + Construtores da RAF:
    - Recebem como parâmetro o nome e o modo

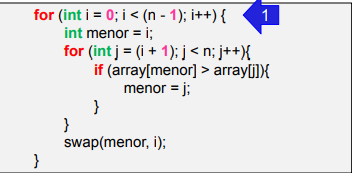
EX:

* + EX:
  + Não é estático.
  + Metodos:
* Redirecionamento de entrada e saída
  + Redirecionamos a entrada ou saída em tempo de execução usando os sinais de < e >.



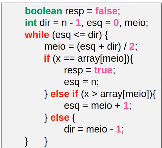
Aula dia 10/08/2022

**Continuação Algoritmos de Ordenação Interna UNIDADE1 - D**

* Existe alguns casos que você deve trocar o elemento com ele mesmo, pois é mais barato que comparar elementos entre arrays
* ****O laco externo de seleção executa para os valores de i entre 0 e n-2. O i não precisa chegar na posição n-1, pois essa terá o maior elemento. Esse é o motivo dos limites do seleção (número de comparações) serem ∑(n-2)^ e 0
* Neste caso o for interno serve para achar o menor e o externo para trocar.
* Analise dos números de movimentações e comparações
  + Executamos o laco interno n – i + 1.
  + Comparação: ∑(n-i+1) i=0 até n-2 àC(n) = n^2 – n/2 = Ɵn^2
  + Número total de movimentação entre elementos do array 3(n-1)

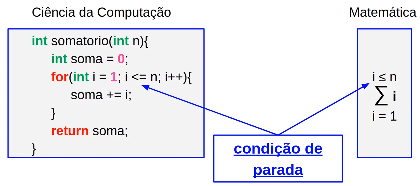
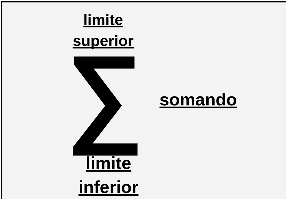
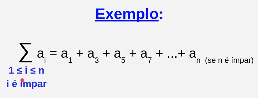
Estudo Individual 11/08/2022 + Aula 17/08/2022 – Somatórios Introdução Max

**Video - Algoritmos de Pesquisa 01**

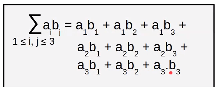
* Pesquisa sequencial:
  + Verificar cada uma das posições do vetor.
  + Melhor caso = 1
  + Pior caso = n
* Pesquisa binária
  + O vetor deve estar ordenado
  + Diminui o escopo de busca pela metade – com uma comparação
  + Melhor caso – 1
  + Pior caso – 2 \* lg(n) .

**SOMATÓRIOS**

**Video - Somatórios**

* Motivação:
  + Levantamento de custo de algoritmos
  + O custo de um algoritmo é a soma dos custos de sua operação
* Notação:
  + OBS:limites podem ser representados em baixo
  + P(i) à propriedade que restringe a soma ex : some apenas os impares

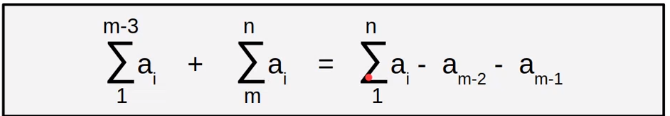
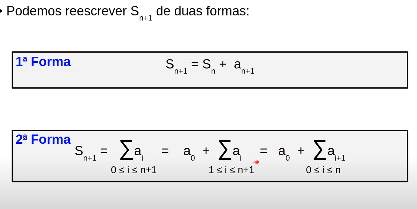
**Relações de Recorrência e Somas Múltiplas**

* Técnica para calcular somas
* Sequência de Fibonacci
* ****Os termos de um somatório podem ser especificados por dois os mais índices EX:

**Somatórios 04 Manipulação de Somas Regras Básicas de Transformação**

* Habilidade de transformar uma soma em outras mais simples
* Regras básicas de Transformação:
  + Distributividade
    - Permite mover constantes para dentro ou fora do somatório
    - ****
    - ****
    - Serve pra divisão
  + Associatividade
    - Permite quebrar um somatório em partes os unifica-las em um somatório
    - ****
    - ****
    - Serve para subtração e adição
    - Mesmo limites
  + Comutatividade
    - ****
    - ****
    - Ordem não importa.

**Somatórios Manipulação de Somas Propriedade P1 e P2**

* Propriedades:
  + P1: Combinando conjuntos
    - Combina conjuntos de índices diferentes.
    - 
    - União + interseção
    - 
  + P2:Base para a Perturbação
    - 
    - 

**Métodos Gerais**

* Procure
* Prove por indução
  + Passo base = Igular n ao primeiro termo
  + Indução 

Aula dia 16/08/2022 - laboratório

* Em Java pode-se ter uma função om o mesmo nome na mesma classe com diferentes parâmetros (encapsulamento de acesso).

Aula dia 22/08/2022

* Quando temos um somatório onde será o valor máximo – i estamos simulando um for decrescente.
* Somatório de Gaus = n\*(n+1)/2 começa em 0 e vai até n.
  + Quando termina diferente termos que substituir n pelo o novo n
* Somatório de 1 começa em 0 e vai até n = (n+1)
* Na cola para prova lembrar de colocar a formula da P2.

Aula dia 24/08/2022

* ∑i^2 à precisamos fazer o ao cubo para resolver
* Indução propriamente dita:

Aula 29/08/2022 – **UNIDADE 3 – FUNDAMENTOS DE ANÁLISE DE ALGORITIMOS**

* Método de seleção por inserção (FOTO)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| G | A | B | R | I | E | L |

* + Compara com a casa anterior, caso não ordenado e insere a nova letra
  + Melhor caso(quando ordenado): n-1(linear)
  + Pior caso: n^2
* Melhor tempo de todos = n\*log(n) – caso médio (não se conta o melhor caso)
* Função de complexidade da o custo do algoritmo
* Algoritimos NP – e^n – problema do caixeiro viajante à intratáveis

Estudo Individual 30/08/2022

**Microfundamento: Algoritmos e Estrutura de Dados – UNIDADE 2 – Introducao + Lista Linear**

* Dois atributos: array e contador.
* Listas podem ser flexíveis e lineares
  + Linear: contador até máxima capacidade do array e depois a proibição de novos inserções + armazenamento sequencial
  + Flexivel: quando atingimos a capacidade de armazenamento do array, cria um novo array (maior que o atual), copia todos os elementos do array antigo para o novo e desaloca o antigo + armazenamento de pares valor/referencia
* Pilhas x fila (Ordem de saída):
  + Na pilha, o primeiro elemento que entra é o último a sair e o último que entra é o primeiro a sair.
  + Na fila, o primeiro elemento que entra é o primeiro a sair
* Lista linear:
  + Métodos InserirInicio, InserirFim(da lista), Inserir, RemoverIicio, RemoverFim, Remover e Mostrar.
  + Sempre teremos que pensar na execao :
  + Pode ser modificado
  + Se o array estiver preenchido com sua máximo tamanho será impossível fazer uma nova inserção de elemento. – obs ArrayList irá dobrar o tamanho.
  + Remover à verificar apenas se n !=0
  + Não removemos fisicamente apenas logicamente.
  + Podemos alterar essa lista para criar:
    - Pilha: é uma implementação onde a inserção e a remoção acontecem no fim da estrutura.
    - Fila: A fila é similar a lista, tendo apenas os métodos de InserirFime RemoverInicio
    - Lista ordenada: A lista ordenada deve ter apenas um método de inserção que garante que os elementos serão inseridos na ordem desejada
    - Deque: O deque é uma estrutura que gerencia inserções e remoções nas duas extremidades do array. O deque tem um contador para gerenciar os elementos inseridos no início do array e outro para os no fim. O somatório desses contadores sempre será menor ou igual a capacidade do array.

Aula dia 31/08/2022

* Ɵ = Limite justo – tem quer ser Ω e O
* O=Limite superior - será todos os valores maiores também
* Ω=Limite inferior – será todos os valores menores também

Aula dia 05/09/2022 – **UNIDADE 4 - A TIPOS ABSTRATO DE DADOS LINEARES - LISTA**

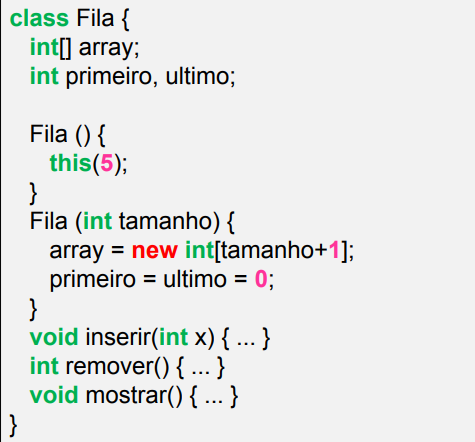
* A listas são um tipo abstrato de dados no qual podemos inserir e removomer elementos em **qualquer posição. EX:** lista de valores, nomes, notas , carros
* Métodos InserirInicio, InserirFim(da lista), InserirQualquerPosicao, RemoverIicio, RemoverFim, RemoverQualquerposicao e Mostrar.
* Sempre na construção de listas temos que pensar nas exceções e assim no tratamento das mesmas
* N à será quantos elementos tem no seu array, ou seja, referência de deslocamento

**UNIDADE 4 - B TIPOS ABSTRATO DE DADOS LINEARES – PILHA**

* As pilhas são um tipo Abstrato de dados no qual o primeiro elemento que entra é o último a sair
* FILO(First in last out)
* Metodos: Inserir(empilhar,push) remover(desempilhar,remove)
* Só podemos ter a opção de II e RI ou IF e RF

Aula dia 12/09/2022 - **FILA**

* As filas são um tipo abstrato de dados no qual o primeiro elemento que entra é o primeiro a sair
* Fifo(First in first out)
* Métodos: Inserir e remover (IF e RI OU II e RF)
* Inserção – Ɵ(n) – Remoção Ɵ(1) ou – Ɵ(1) – Remoção Ɵ(n)
* Para fazer uma fila sem deslocamento – Fila circular ou seja, depois da ultima posição retornamos para primeira
* Usamos % à para fazer a fila circular
* Algoritmo em Java:



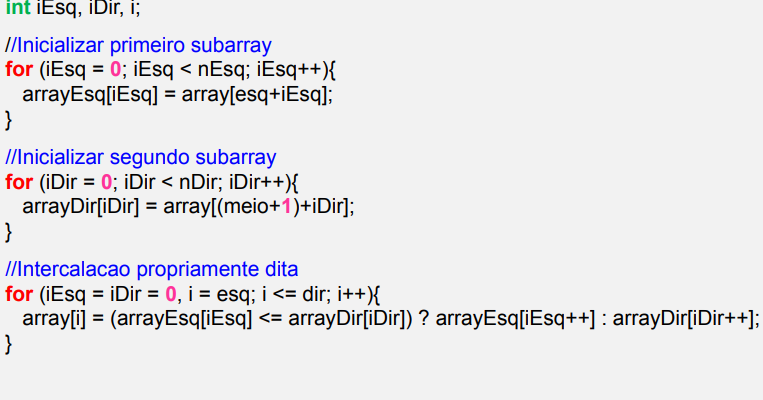
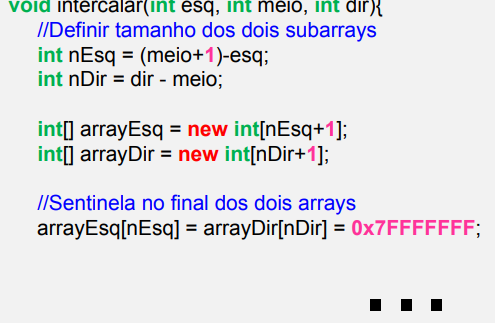
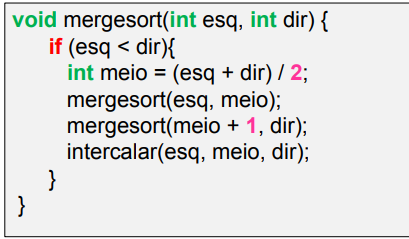
Aula dia 21/09/2022

* QuickSort
  + Divide array em duas partes que serão independentemente ordenados e a combinação de deus resultados será a solução final
  + A parte da esquerda terá elementos menores ou iguais a um pivô
  + A parte da direita terá elementos maiores ou iguais a um pivô
  + Logica:
    - Escolha arbitrariamente um pivô
    - Percorra o array a partir da esquerda enquanto array[ i ] < pivô
    - Percorra o array a partir da direita enquanto array[ j ] > pivô
    - Se i ≤ j então troque array[ i ] com array[ j ]
    - Continue o processo enquanto i ≤ j
  + Melhor caso : N . LOG N
  + Pior caso: N^2 – quando a escolhas do pivô pega o maior ou pior

Aula dia 26/09/2022

* Continuação QuickSort
  + Para evitar o pior caso existem passos que podemos fazer.
  + A escolha ideal é a mediana.
  + Escolha do pivô:
    - Mediana depende da ordenação
    - Quando conhecemos os dados , podemos utilizar a estratégia de escolhas do pivô mais adequados a situação
    - Técnica de mediana de três elementos aleatórios.
    - Técnica do primeiro ou ultimo elemento
    - Técnica do elemento do meio, em que o pior caso do Quick acontece se elementos formarem um triangulo
  + A razão de sua velocidade é a simplicidade do seu anel interno:
    - Extremamente eficiente
    - Necessita apenas de uma pequena pilha como memoria auxiliar’
  + Desvatagens:
    - Pior caso n^2
    - Sua implementação é delicada e dificl
    - Método não estável

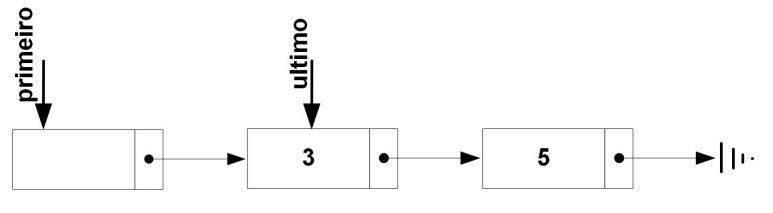
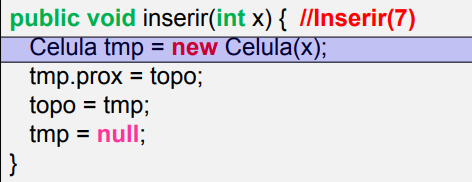
Aula dia 28/09/2022 - **Merge Sort**

* Intercalações de vetores ordenados:
  + Premissa: vetor ordenado
  + Merge Sort: ordenação através de intercalações de vetores
    - Gasta mais memoria, porém SEMPRE será (n \* log n)
    - Algoritmo de ordenação do tipo dividir para conquistar
    - Implementação de forma recursiva e demanda um espaço adicional de memória(não é um algoritmo **inplace**)àinplace = se a memória adicional requerida é independente do tamanho do array
  + Funcionamento:
    - Dividir o array em subarrays até que tenham tam = 1
    - Intercalação de dois subarrays
    - Pilha de recursão
    - Sentinela serve para marcar limite de array para evitar erro de percorrer array à economizar if
  + Analise de comparações
    - Em cada subarray fazemos k-1 comparacoes
    - Lg(n) passos, pois esta dividido
    - Ou seja, N \* lg(n)
  + Analise de movimentações
    - N\*LG(N)
  + Conclusão
    - Estável
    - Não é inplace
    - Forma recursiva + memória adicional
    - Caso médio, pior e melhor = n \* lg(N)

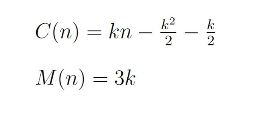
Aula dia 03/10/2022 – Início da Unidade 6 + Revisão de ponteiros

* Int [] Y = new int [5] à y aponta para uma estrutura de array, onde o Y aponta para uma POSIÇÃO e não um elemento.
* Ponteiro do JAVA é implícito e passagem por referência.
* Site para estudos: phythortutor.com.br
* Se existem dois ponteiros apontando pro mesmo lugar, qualquer mudança em um irá mudar para os outros, pois eles apontam para o mesmo lugar na memóriaà solução: **usar método clone!**

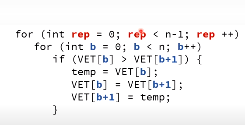
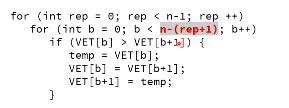
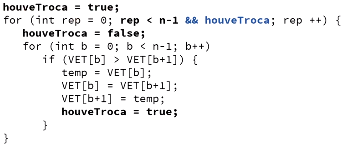
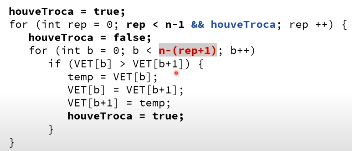
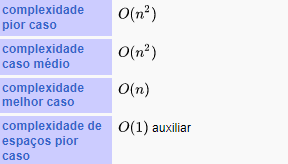
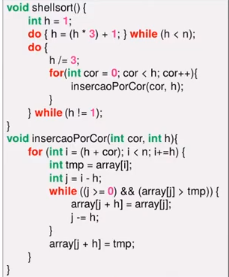
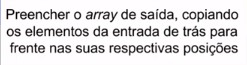
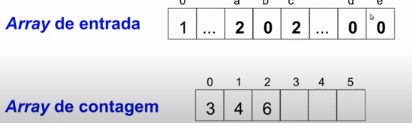
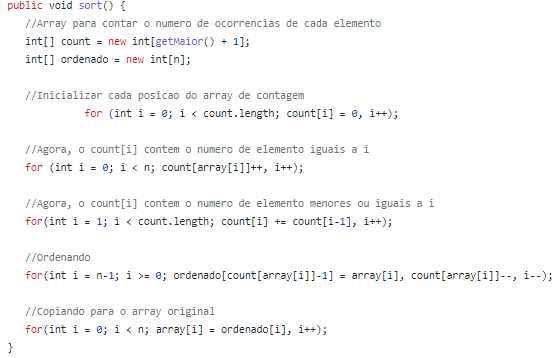
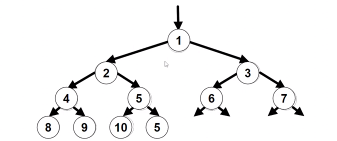
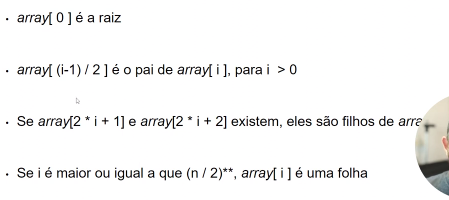
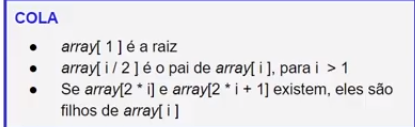
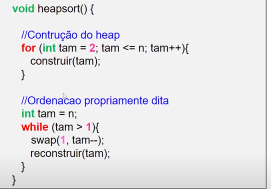
Tipos abstratos de dados:

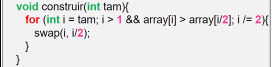
* Lista flexível:
  + Lista com ponteiro
  + Boa pratica: atrbibutos privados – GET E SETTERS
* Pilha: - **INSERE E TIRA DO TOPO**
  + Vazio : aponta pra null
  + Insere: insere e aponta pra nulo , porém o topo aponta para nova célula criada.
  + Topo sempre irá apontar para uma célula e o ultimo será nulo.
  + Remoção:
    - Testa se está vazia
    - Pega conteúdo do topo para retornar
    - Cria célula temporária
    - Muda topo pra o próximo
    - Muda temp pra null - quebra ligação
  + Mostrar:
    - Looping até i = null

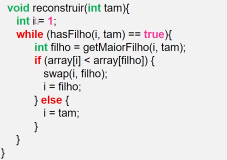
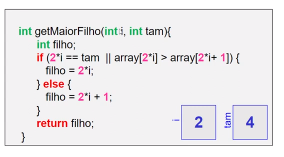
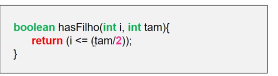
Aula dia 04/10/2022

* Ordenação parcial K elementos
  + Ordenar apenas alguns elementos
  + K irá variar de k até n
  + Nova complexidade:

Estudo individual Unidade 5 04/10/2022

* **Método da bolha**:
  + Solta a bolha na posição 0 e varre o vetor para achar o maior e joga para o fim.
  + Também pode ser invertido achando o menor e jogando para o fim do array
  + Comparações: n-1 a cada interação
  + Interacoes : n-1
  + Bolha sempre chegará até a **penúltima** posição , pois não existe o elemento seguinte
  + Melhor caso: quando o vetor está ordenado – n(n-1)/2
  + Pior caso: ordenado em ordem decrescente – 3 \* n(n-1)/2
  + Algoritmo:
  + Adaptacoes para melhoria de performace:
    - É possivel reduzir a quantidade de comparacoes a cada iteração, pois o maior elemento já esta na posicao final:
    - Redução no número de iterações – verificar ordenações a partir das trocas, caso não tenha trocas já está ordenado:
    - Combinação das duas:
    - RESUMO: .
    - Estavel
* **Shellsort**
  + Proposto por Shell como uma extensão do Algoritimo de Inserção
  + O problema da Inserção acontece quando o novo elemto é menor que os demais, causando i-1 comparacoes e movimentacoes
  + Funcionamento básico:
    - Dividimos em h grupos(separados de h posicoes) e ordenamos casa um dele usando o Inserçao
    - Neste momento o array esta h-ordenado
    - Em seguida, reduzimos o h e repetimos o processo até h = 1.
  + Sequencia de passos:
    - Não se sabe qual a melhor sequência
    - Se usa 1,4,13,40,121 – 3h(s-1)+1
  + Algoritmo em c like:
    - Inserçao , porém tudo que for 1 trocamos por h, pois andamos de h em h
  + Complexidade e Conclusão:
    - Razao da eficiência ainda não é conhecida
    - Cada incrementado não deve ser múltiplo do anterior
    - Vantagens:
      * Otima opção para arquivos de tamanho moderado
      * Simples e requer pouco codigo
    - Desvantagens
      * Seu tempo de execução é sensível a ordem inicial do arquivo
      * Algoritmo não estável
* **CountingSort**
  + Introdução, Ideia básica e Exemplo:
    - Triplicamos o numero de arrays(entrada,contagem,saída)
    - Array contagemàcada posição do contagem armazena o número de elementos menores ou iguais a ela no entrada.EX:
    - Array de saída à array ordenado
  + Analise de complexidade:
    - Operaçoes à 4 \* O(n)àO(n) custo linear
  + Algoritimo em c like:
  + Várias limitações: float...
* **HeapSort**
  + Algoritimo de seleção que encontra o maior elemento e troca com o ultimo, repetindo o processo
  + Diferença que usa um heap invertido(heap máximo) para selecionar o maior elento de forma eficiente
  + Definição de Heap
    - **Árvore:**
      * Estruturas de dados cujas operações inserção, remoção, substituição possuem a mesma eficiência.
      * Contem um conjunto finito de vértices(nos) e outro de arcos(arestas) que conectam os vértices
      * Raiz à nó especial(Não tem pai)
      * Nó com filhos = internos
      * No sem filhos = folhas
      * Arvore binária à cada nó tem no máximo dois filhos(0,1,2)
    - **Heap:**
      * Arvoré binária em que cada no é menor ou igual que seus filhos , fazendo sua raiz o menor valor
      * Suas folhas ocupam um ou dois níveis sendo que o penúltimo é completo e as folhas do ultimo se agrupam a mais esquerda possível   
        EX:
    - **Heap Invertido**:
      * Raiz guarda o maior valor da arvore, sendo que cada nó é maior ou igual que seus filhos
      * Suas folhas ocupam um ou dois níveis sendo que o penúltimo é completo e folhas do ultimo nível é a mais esquerda possível
    - Heap em array(índice 1 e 0):
    - 
    - Operações em um Heap:
      * Construção
      * Inserção de um novo elemento
      * Remoção do maior
      * Remoção de qualquerUM
      * Unificar heaps
  + Funcionamento básico:
    - Construção um heap inserindo sistematicamente elementos
    - Remover elemento de maior valor + reconstrução do heap+insrir elemento no array ordenado
    - Principio da inserção do heap:
      * Crie uma nova folha no ultimo nível do heap.
      * Se o novo elemento for maior q o seu pai troque-o e realize o mesmo processo
    - Principio da remoção do heap:
      * Armazene o elemento da raiz em uma variável temp
      * Substitua o elemento da raiz pelo da ultima folha do ultimo nível
      * Remova a ultima folha do último nível
      * Troque o elemento da raiz com seu maior filho
      * Repita o passo anterior para o filho com elemento trocado até que todos pais sejam maiores que seus filhos.
  + Algoritimo c like:
  + Construir – inserção



* + Reconstruir – remocao
  + getMaiorFilho
  + hasFilho:
* Analise de comparações e movimentações:
  + Comparações no pior caso: 2n log2n + O(n) é o mesmo que 2n log n + O(n)
  + Trocas no pior caso: n log2n + O(n) é o mesmo que n log n + O(n)
  + Melhor e pior caso: O(n log2n) é o mesmo que O(n log n)

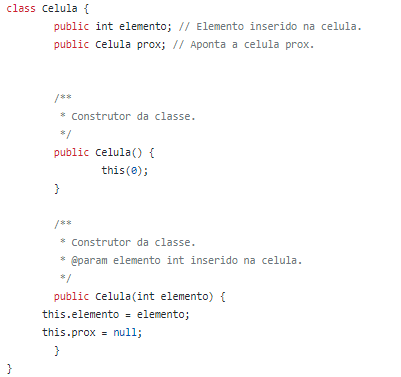
Classes autorreferenciáveis:

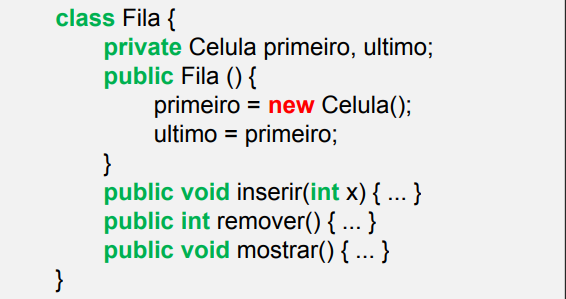
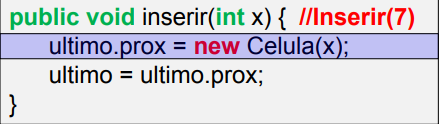
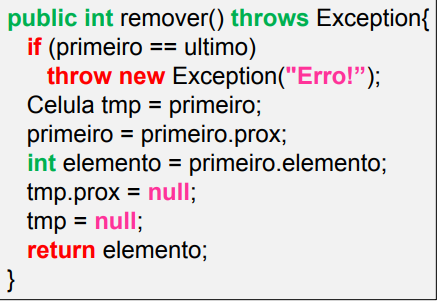
* No c# e java nos não trabalhamos diretamente com objetos e sim com referências para eles
* Classes autorreferenciais possuem uma referencia para objetos de seu tipo, referência para ela mesma

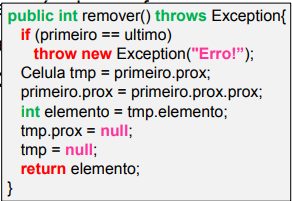
Aula dia 05/10/2022

Tipos Abstrados de dados flexíveis – Fila – FIRST IN FIRST OUT – INSERE FIM E REMOVE INICIO

OBS: Celula:

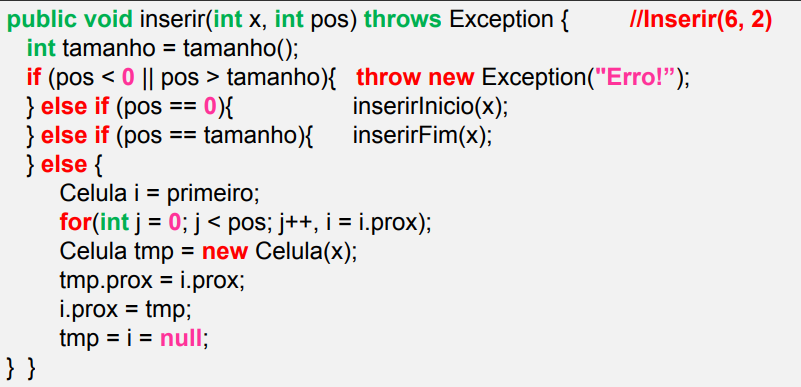
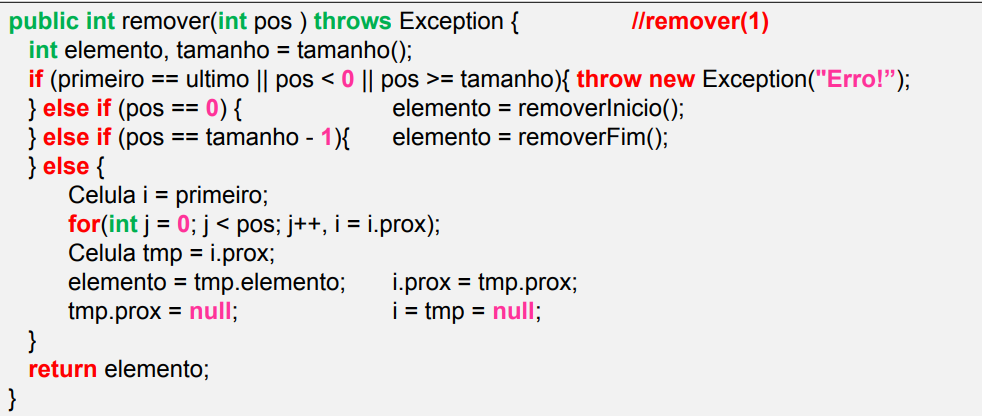


* Precisaremos **marcar o inicio e o fim**
* Inserir – **NO FINAL:**
* Remover – **NO INICIO: - Remove cabeça e o próximo vira a cabeça**
* Remover – **NO INICIO – Muda o ponteiro do cabeça**

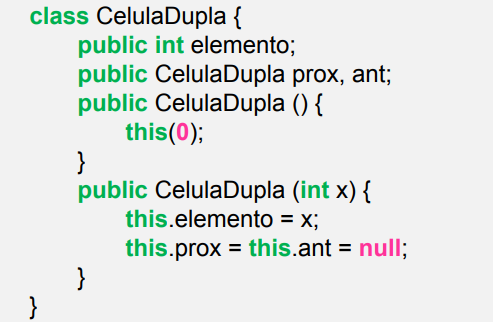


* OBS:
  + Criamos uma célula primeiro vazia de referencia – economizar o if para testar o null

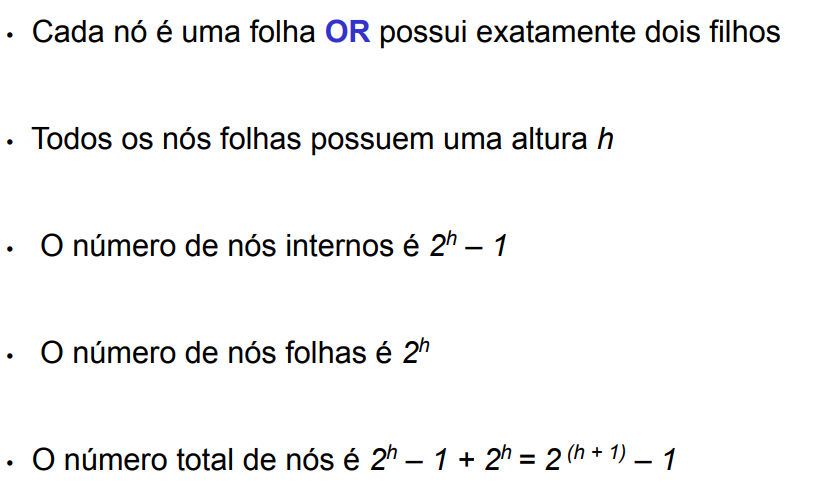
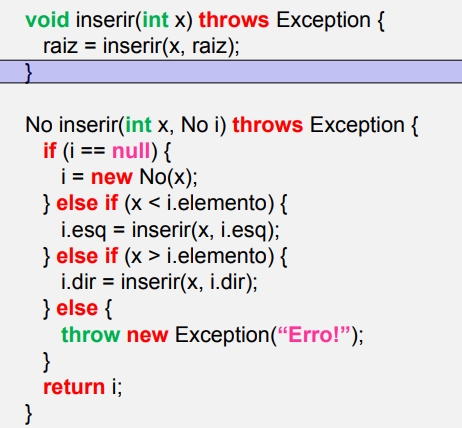
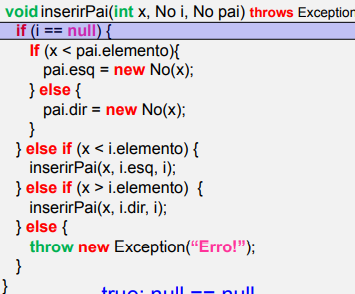
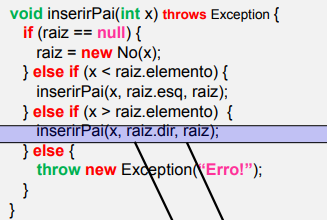
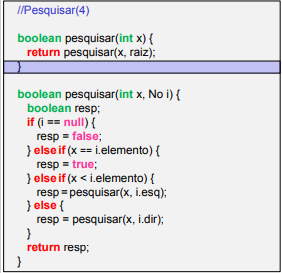
**LISTA SIMPLES:**

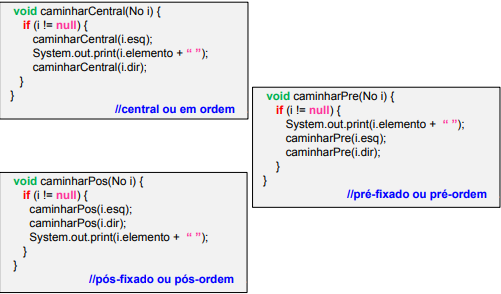
* Juncao fila e pilha + inserirPosicao e removerPosicao
* InserirPosicao:
* RemoverPosicao:

Aula do dia 17/10/2022 – Lista dupla

* Necessário da célula dupla – dois ponteiros
* Mariz
  + Nossa celula terá 4 apontadores – esq dir inf sup

Aula dia 24/10/2022 – Arvore Binaria - i

* Raiz à topo – maior elemento
* Folha à Não tem filho
* No interno à no com filho
* Altura(h) é a maior distância entre um nó e a raiz.
* Características:
* Arvore balanceada: Árvore em que para TODOS os nós, a diferença entre a altura de suas árvores da esquerda e da direita sempre será 0 ou ±1 .
* Inserção:
  + Pode ser com retorno por referencia : 
  + Pode ser com passagem de pai:
  + 
* Complexidade:
  + MC = 1
  + CM = lg(n)
  + PC = n
* Pesquisa e Caminhamento:
  + Funcionamento básico:
    - Se a raiz estiver vazia, retornar uma pesquisa negativa
    - (2) Senão, se o elemento procurado for igual ao da raiz, retornar uma pesquisa positiva
    - (3) Senão, se o elemento procurado for menor que o da raiz, chamar o método de pesquisa para a subárvore da esquerda
    - (4) Senão (elemento procurado é maior que o da raiz), chamar o método de pesquisa para a subárvore da direita
  + Algoritimo:
  + Complexidade:
    - MC = 1 – Quando for a raiz
    - CM = lg(n) – Arvore toda caído pra um lado
    - PC = n – Esta na folha ou não existe
* Caminhamento

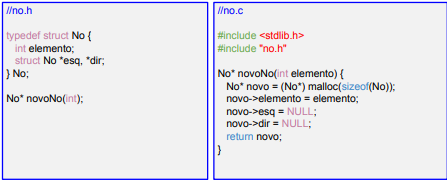
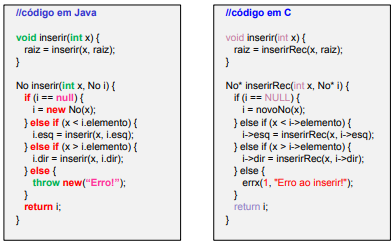


* + Complexidade - - n
  + Tres tipos – so muda o momento da impressao
    - Central – em ordem
    - Pre – razi esquerda direita PRIMEIRO E A RAIZ
    - Pos- esquerda direita raiz – ULTIMO E A RAIZ

Aula dia 25/10/2022

* Java tem um coletor de lixo que facilita o gerenciamento de memoria, diferentemente de C que precisamos dar free
* Metodo finalize à responsável pela limpeza de memoria,

Aula dia 26/10/2022 – Arvore binaria – inserção em c com ponteiro

* Ponteiros:
  + & = endereço da variável
  + \* = conteúdo da posição de memória endereçada pelo ponteiro
* Diferença de \*x e x\*:
  + \*x à ponteiro
  + X\*à multiplicação de x
* Sempre usar malloc e free
* MakeFile:
  + Arquivo contendo um conjunto de diretias usadas pela ferramenta de automação de compilação make para gerar um alvo/meta
  + Nesse caso os arquivos serão compilados digitando make
  + Junção de arquivos que depende dos outros – precedência de compilação
* Arquivos nó :
* Inserção:

Aula dia 31/10/2022 - Arvore binaria – inserção em c com ponteiro CONTINUACAO + Unidade vii – C++

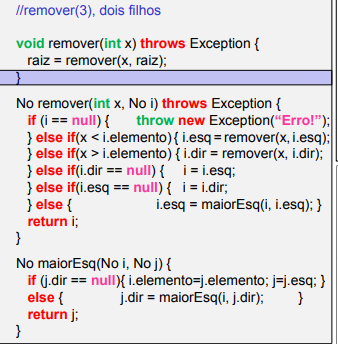
* Tomar cuidado com o ponteiro de ponteiro
* Não existe passagem por referencia e, JAVA apenas em C e C++
* Problema da árvore em C++ : passagem de parâmetro
  + Passagem por referência em C++:
    - Passamos uma referência (apelido), fazendo com que qualquer alteração no método chamado seja refletida no que chama
    - Void função(int&a,int b)
    - No swap não há troca de apontamento e sim de conteúdo
    - Diferença da passagem por referência usando & e \*:
      * Não tem diferença apenas para pegar o valor e atribuição
      * Quando usamos void função (&a,\*b):
        + Mandamos a e &b
        + Usamos a e \*b para recuperar o valor
* Referência x manipulação de ponteiros em c++:
  + Int \* p = & x -- ponteiro
  + Int & r = x -- referencia

Estudo individual 03/11/2022

**Arvore Binaria**

* Custo de inserção , remocao e pesquisa nas listas = O(n)
* Nas arvores, pode ser O(lg(n))
* **Raiz** – esta no nível 0 , primeiro elemento
* **No interno** é aquele que tem filhos
* **Na folha/externos** é aquele que NÃO tem filhos
* **Altura =** maior distancia entre um nó e a raiz
* **Arvore binaria de pesquisa:** cada nó é maior que todos vizinhos do lado esquerdo e menor do lado direito
* **Arvore balanceada:** a diferença entre a altura de suas árvores da direita e esquerda sempre será 0 ou +-1

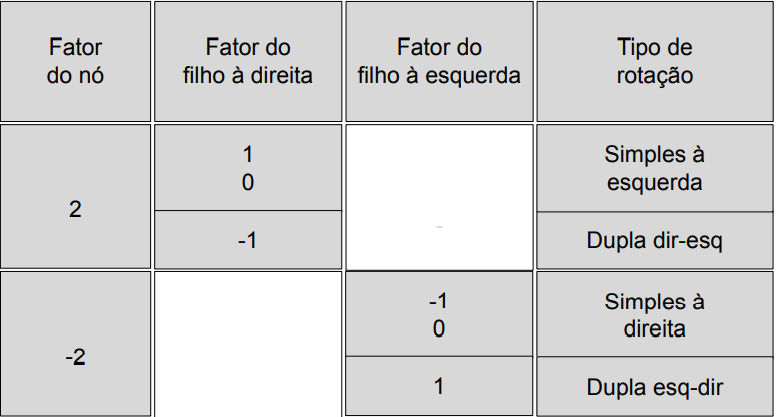
Aula dia 14/11

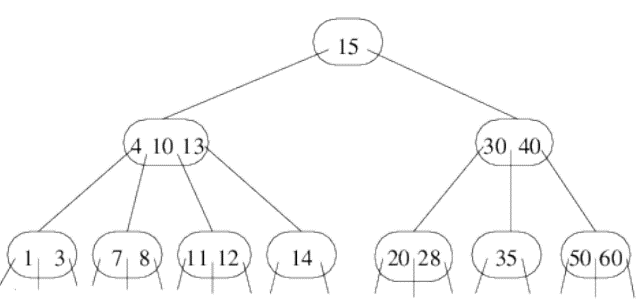
* Para remoção temos que pensar qual será o elemento que irá substituir o removido (3 cenários)
  + Se o elemento estiver uma folha, remove-la
  + Se o elemento estiver em um no intermediário com um filho, remover o no e fazer com que seu pai aponte para seu filho.
  + Se o elemento estiver em um nó intermediário com dois filhos, o elemento a ser removido deve ser substituído ou pelo maior nó da subárvore à esquerda ou menor nó da subárvore à direita.
* ****Algoritmo:
* Complexidade:
  + MC – 1
  + PC n
  + CM lg n

Aula dia 16/11/2022 – Balanceamento de arvore

* Fator desbalanceamento: AlturaDireita – AlturaEsquerda – cada nó tem o seu
* Balaceada 🡪 -1<=FATOR<=1
* Se um nó estiver desbalanceado 🡪 **árvore desbalanceada**
* Duas propostas de implementação:
  + Atributo: Nível – mais difícil de calcular balanceamento porem n precisa atualizar toda hora
  + Atributo : Fator – mais fácil de calcular balanceamento porem precisa atualizar
* Como balancear:
  + Rotação simples a esquerda:
    - Usada em subárvores em que o pai e o filho estão desbalanceados para a direita. FB > 0
  + Rotação simples a direita:
    - Usada em subárvores em que o pai e o filho estão desbalanceados para a esquerda. FB < 0
  + Rotacao dupla a direita – esquerda:
    - Usada em subárvores em que o pai está desbalanceado para a direita e o filho para a esquerda FB > 0 e FB > 0 filho
  + Rotacao dupla a esquerda – direita
    - Usada em subárvores em que o pai está desbalanceado para a esquerda e o filho para a direita FB < 0 e FB < 0 filho

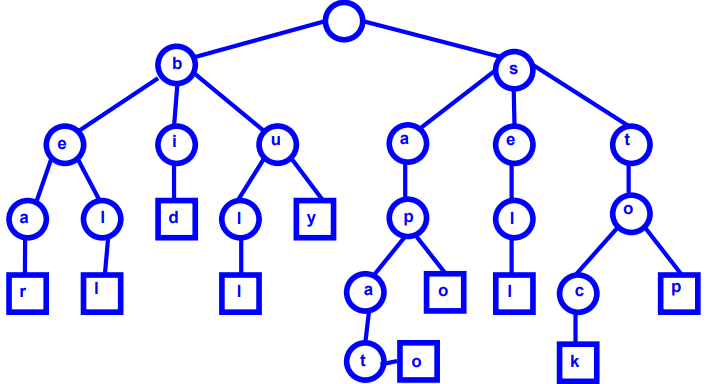
Aula dia 21/11/2022 – Arvoré AVL

* Sempre será balanceado, pois a cada inserção e remoção será balanceada.

Aula dia 23/11/2022 – Arvore 2-3-4

* Regras:
  + Folhas estão sempre no mesmo nível 🡪 balanceada
  + Array terá no máximo 3 elementos
  + Menores a esquerda e maiores a direita
* Pesquisa:
  + Verificar se o elemento procurado x está no nó raiz
  + (2) Se estiver, tem-se uma resposta positiva
  + (3) Senão, se x < x1, verificar na subárvore da esquerda
  + (4) Senão, se x < x2, na do meio à esquerda
  + (5) Senão, se x < x3, na do meio à direita
  + (6) Senão, na da direita
  + (7) Se a subárvore for nula, tem-se uma resposta negativa – ou seja n achou
  + MC – (1)
  + CM e PC (lg(n))
* Inserção pela fragmentação na subida:
  + Quando a raiz n caber vc sobe o elemento do meio e quebra o array
* Insercao pela fragmentação na descida:

Aula dia 05/12/2022 – Arvore TRIE

* Retrival – recuperação -- tabela de string, visando o prefixo igual , buscando por padrões
* Estrutura de dados para procura rápida de padrões – **necessita de um dicionário**, conjunto de caracteres validos
* Pode-se ter n filhos
* Raiz sempre será vazio
* EX:

Aula dia 07/12/2022

* A letra final da chave de pesquisa é marcada , por exemplo com uma variável booleana
* Na remocao se não for prefixo de nada será removido , porém se for so sera desmarcado