ADS 6.0

Teste de Caixa-Branca

Integrantes:  
  
  
Kauã Freitas Santana Silva(ADS) RA: 823116576

Carolina dos Santos Araujo(ADS) RA: 82324786

Jonatan Mateus de Souza Alves(ADS) RA: 823121880

Letícia Girardi Marra(ADS) RA:823126898

Lorena Oliveira Solaci(ADS) RA: 823112471  
  
Rafael O. de Carvalho Mendes(ADS) RA: 823128825

Introdução  
  
  
Para realizar o teste de caixa-branca sobre o método fornecido, vamos focar em avaliar os caminhos de execução do código. O teste de caixa-branca examina a lógica interna e a estrutura do código, garantindo que todas as partes dele sejam exercitadas adequadamente**.** Testamos o código dado pelo professor, notamos que faltavam alguns ajustes, existiam alguns erros de digitação, então arrumamos e abaixo deixaremos o código da maneira correta:  
  
***public static int busca\_binaria(int iVet[], int iK) {***

***int iBaixo, iAlto, iMeio;***

***iBaixo = 0;***

***iAlto = iVet.length - 1;***

***while (iBaixo <= iAlto) {***

***iMeio = (iBaixo + iAlto) / 2;***

***if (iK < iVet[iMeio]) {***

***iAlto = iMeio - 1;***

***} else if (iK > iVet[iMeio]) {***

***iBaixo = iMeio + 1;***

***} else {***

***return iMeio; // elemento encontrado***

***}***

***}***

***return -1; // elemento não encontrado***

***}***  
  
Agora que o código está corrigido, podemos realizar a análise de teste de caixa-branca!!!

# Análise do Fluxo de Controle Vamos identificar os caminhos possíveis que o fluxo de controle do código pode seguir, considerando as diferentes condições.

#### 1 – Análise do Fluxo de Controle: Um laço while com a condição iBaixo <= iAlto. Três condições no bloco if-else if-else: iK < iVet[iMeio]: Reduz o limite superior (iAlto). iK > iVet[iMeio]: Aumenta o limite inferior (iBaixo). iK == iVet[iMeio]: Retorna o índice onde o valor foi encontrado. 2 – Caminhos Lógicos e Casos de Teste: Aqui estão os 4 caminhos que devemos testar para cobrir todos os caminhos do código, sem deixar passar nada: Caminho 1: O valor está no meio do vetor (caso ótimo)

### Neste caminho, o elemento buscado será encontrado na primeira iteração do loop.

#### Exemplo: Para o vetor {1, 2, 3, 4, 5}, buscar iK = 3 resultará na primeira iteração com iMeio = 2 e retornará 2. Caminho 2: O valor está na metade inferior do vetor

O elemento buscado estará em uma posição inferior à posição média, fazendo com que o código atualize iAlto.

#### Exemplo: Para o vetor {1, 2, 3, 4, 5}, buscar iK = 1 resultará em uma primeira iteração com iMeio = 2. Como iK < iVet[iMeio], o código ajustará iAlto para 1. O laço continuará até encontrar o valor na segunda iteração.

Caminho 3:O valor está na metade superior do vetor

### O elemento buscado estará em uma posição superior à posição média, fazendo com que o código atualize iBaixo.

#### Exemplo: Para o vetor {1, 2, 3, 4, 5}, buscar iK = 5 resultará em uma primeira iteração com iMeio = 2. Como iK > iVet[iMeio], o código ajustará iBaixo para 3. O laço continuará até encontrar o valor na segunda iteração.

#### Caminho 4: O valor não está presente no vetor

### Neste caso, o valor buscado não existe no vetor e o laço terminará sem encontrar o valor.

#### Exemplo: Para o vetor {1, 2, 3, 4, 5}, buscar iK = 6 resultará em várias iterações, com iBaixo sendo atualizado até ultrapassar iAlto, e o método retornará -1.

### 

### 3 - Casos de Teste Derivados: Com base nos caminhos identificados, os seguintes casos de teste devem ser realizados: 4 - Conclusão A abordagem de teste de caixa-branca se concentra em garantir que todas as ramificações do código (caminhos possíveis) sejam testadas. Para este algoritmo de busca binária, os principais caminhos incluem:

### 1. Encontrar o valor na primeira tentativa.

### 2. Ajustar o intervalo de busca para a metade inferior.

### 3. Ajustar o intervalo de busca para a metade superior.

### 4. Caso o valor não seja encontrado no vetor.

### Executando esses testes, garantimos que o método de busca binária funciona corretamente em diferentes cenários.

##### 