

LPG0001 – Linguagem de Programação

Vetores, Funções, Algoritmos de Busca

Prof. Rui Jorge Tramontin Junior
Departamento de Ciência da Computação
UDESC / Joinville

Introdução

- Vetores podem ser passados como parâmetros para funções;

Introdução

- Vetores podem ser passados como parâmetros para funções;
- Ao contrário de variáveis simples, os valores de um vetor podem ser modificados dentro de uma função;
 - Vetores são sempre **passados por referência!**

Introdução

- Vetores podem ser passados como parâmetros para funções;
- Ao contrário de variáveis simples, os valores de um vetor podem ser modificados dentro de uma função;
 - Vetores são sempre *passados por referência!*
- Normalmente passa-se também a capacidade do vetor como parâmetro da função.

Exemplo 1: média dos valores

```
// programa principal

#define MAX 10

int main(){
    float v[MAX] = {6, 9, 4, -3, 17, 22, 13, 81, 44, 32};

    float media = calcula_media (v, MAX);

    printf("Média: %f\n", media);

    return 0;
}
```

Exemplo 1: média dos valores

```
float calcula_media(float vet[], int n){  
    int i;  
    float soma = 0;  
    for(i=0; i < n; i++){  
        soma = soma + vet[i];  
    }  
    return soma / n;  
}
```

Exemplo 2: entrada e saída

```
// programa principal

#define MAX 10

int main() {
    int v[MAX];

    le_vetor(v, MAX);
    mostra_vetor(v, MAX);

    return 0;
}
```

Exemplo 2: entrada e saída

```
void le_vetor(int vet[], int n){  
    int i;  
    for(i=0; i < n; i++){  
        printf("Digite o %dº valor: ", i + 1);  
        scanf("%d", &vet[i]);  
    }  
}
```


Exemplo 2: entrada e saída

```
void mostra_vetor(int vet[], int n){  
    int i;  
    printf("Dados do vetor:\n");  
    for(i=0; i < n; i++){  
        printf("%d : %d\n", i, vet[i]);  
    }  
    printf("\n");  
}
```

Exemplo 3: *SelectionSort*

```
// programa principal
```

```
#define MAX 10
```

```
int main() {
```

```
    int v[MAX] = {6, 9, 4, -3, 17, 22, 13, 81, 44, 32};
```

```
    printf("Vetor original:\n");
```

```
    mostra_vetor(v, MAX);
```

```
    ordena_vetor(v, MAX);
```

```
    printf("Vetor ordenado:\n");
```

```
    mostra_vetor(v, MAX);
```

```
    return 0;
```

```
}
```

Exemplo 3: *SelectionSort*

```
void ordena_vetor(int vet[], int n){
    int i, j;
    for(i = 0; i < n-1; i++){
        int i_menor = i;
        for(j = i+1; j < n; j++)
            if( vet[j] < vet[i_menor] )
                i_menor = j;

        int aux = vet[i];
        vet[i] = vet[i_menor];
        vet[i_menor] = aux;
    }
}
```

Busca em um vetor

- Dados um vetor e um valor a ser buscado (*chave* de busca), o algoritmo deve informar:

Busca em um vetor

- Dados um vetor e um valor a ser buscado (*chave* de busca), o algoritmo deve informar:
 - quais o(s) índice(s) onde a *chave* se encontra; ou

Busca em um vetor

- Dados um vetor e um valor a ser buscado (*chave* de busca), o algoritmo deve informar:
 - quais o(s) índice(s) onde a *chave* se encontra; ou
 - que a *chave* não foi encontrada.

Busca em um vetor

- Dados um vetor e um valor a ser buscado (*chave* de busca), o algoritmo deve informar:
 - quais o(s) índice(s) onde a *chave* se encontra; ou
 - que a *chave* não foi encontrada.
- Tipos de Busca: *Sequencial* e *Binária*.

Exemplo 1 : Busca em um vetor (1/3)

- Dado o vetor v:

$v = \{ 6, 9, 4, -3, 17, 22, 13, 81, 44, 32 \}$

Exemplo 1 : Busca em um vetor (2/3)

- Dado o vetor v:

$v = \{ 6, 9, 4, -3, 17, 22, 13, 81, 44, 32 \}$

- Entrada \rightarrow chave = 13

Exemplo 1 : Busca em um vetor (3/3)

- Dado o vetor v:

$v = \{ 6, 9, 4, -3, 17, 22, 13, 81, 44, 32 \}$

- Entrada \rightarrow chave = 13
- Saída \rightarrow “Encontrado no índice 6.”

Exemplo 2 : Busca em um vetor (1/3)

- Dado o vetor v:

$v = \{ 6, 9, 4, -3, 17, 22, 13, 81, 44, 32 \}$

Exemplo 2 : Busca em um vetor (2/3)

- Dado o vetor v:

$v = \{ 6, 9, 4, -3, 17, 22, 13, 81, 44, 32 \}$

- Entrada \rightarrow chave = 20

Exemplo 2 : Busca em um vetor (3/3)

- Dado o vetor v:

$v = \{ 6, 9, 4, -3, 17, 22, 13, 81, 44, 32 \}$

- Entrada \rightarrow chave = 20
- Saída \rightarrow “Valor não encontrado!”

Busca Sequencial

- Pode ser feita em um vetor desordenado;

Busca Sequencial

- Pode ser feita em um vetor desordenado;
- É feito um percurso no vetor, comparando a chave com cada valor do vetor;

Busca Sequencial

- Pode ser feita em um vetor desordenado;
- É feito um percurso no vetor, comparando a chave com cada valor do vetor;
- Possui um desempenho proporcional ao tamanho do vetor (*linear*).
 - Caso o vetor seja 10 vezes maior, o temp de resposta tende a ser também 10 vezes maior.

Busca Sequencial: *main()*

```
// programa principal
```

```
int v[] = {6, 9, 4, -3, 17, 22, 13, 81, 44, 32};  
int k = 10, chave;  
printf("Digite o valor a ser buscado: ");  
scanf("%d", &chave);  
int indice = busca_seq (v, k, chave);  
if(indice == -1)  
    printf("Não encontrado!\n");  
else  
    printf("Encontrado em %d.", indice);
```

Busca Sequencial: Função

```
int busca_seq(int v[], int n, int chave){  
    int i;  
    for(i=0; i < n; i++){  
        if(v[i] == chave){  
            return i; // Encontrado em i!  
        }  
    }  
    return -1; // Não encontrado!  
}
```

Limitação dessa implementação

- Retorna somente a primeira ocorrência da chave;

Limitação dessa implementação

- Retorna somente a primeira ocorrência da chave;
- É possível fazer o código todo na *main()* para lidar com essa limitação; (mas.... sem função!)

Limitação dessa implementação

- Retorna somente a primeira ocorrência da chave;
- É possível fazer o código todo na *main()* para lidar com essa limitação; (mas.... sem função!)
- No futuro, veremos como retornar um vetor contendo os índices onde a chave se encontra.
 - Ponteiros (para alocar um novo vetor);
 - Passagem por referência (tamanho do novo vetor).

Busca Binária

- Só funciona em um vetor ordenado!

Busca Binária

- Só funciona em um vetor ordenado!
- Dado o espaço de busca entre os índices 0 e $n-1$:

Busca Binária

- Só funciona em um vetor ordenado!
- Dado o espaço de busca entre os índices 0 e $n-1$:
 - Determina-se o índice médio do vetor;

Busca Binária

- Só funciona em um vetor ordenado!
- Dado o espaço de busca entre os índices 0 e $n-1$:
 - Determina-se o índice médio do vetor;
 - Verifica-se se a chave está antes ou depois;

Busca Binária

- Só funciona em um vetor ordenado!
- Dado o espaço de busca entre os índices 0 e $n-1$:
 - Determina-se o índice médio do vetor;
 - Verifica-se se a chave está antes ou depois;
 - Define-se um novo espaço de busca (metade do tamanho);

Busca Binária

- Só funciona em um vetor ordenado!
- Dado o espaço de busca entre os índices 0 e $n-1$:
 - Determina-se o índice médio do vetor;
 - Verifica-se se a chave está antes ou depois;
 - Define-se um novo espaço de busca (metade do tamanho);
- Desempenho superior ao da busca sequencial.


Busca Binária: Exemplo 1 (1/10)

2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11

- Chave = 90

Busca Binária: Exemplo 1 (2/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 1ª iteração:
 - início = 0
 - fim = 11

Busca Binária: Exemplo 1 (3/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 1ª iteração:
 - início = 0
 - fim = 11
 - meio = 5

Busca Binária: Exemplo 1 (4/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 1ª iteração:
 - início = 0
 - fim = 11
 - meio = 5 → chave está depois do *meio*!

Busca Binária: Exemplo 1 (5/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 2ª iteração:
 - início = 6
 - fim = 11

Busca Binária: Exemplo 1 (6/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 2ª iteração:
 - início = 6
 - fim = 11
 - meio = 8

Busca Binária: Exemplo 1 (7/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 2ª iteração:
 - Início = 6
 - fim = 11
 - meio = 8 → chave está depois do *meio*!

Busca Binária: Exemplo 1 (8/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 3ª iteração:
 - início = 9
 - fim = 11

Busca Binária: Exemplo 1 (9/10)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 3ª iteração:
 - início = 9
 - fim = 11
 - meio = 10

Busca Binária: Exemplo 1 (10/10)

2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 90
- 3ª iteração:
 - início = 9
 - fim = 11
 - meio = 10 → chave encontrada!

Considerações

- O espaço de busca é dividido por 2 a cada iteração;

Considerações

- O espaço de busca é dividido por 2 a cada iteração;
- O número de iterações (tempo de resposta) tende a ser proporcional a $\log_2(n)$:
 - n é a capacidade do vetor;

Considerações

- O espaço de busca é dividido por 2 a cada iteração;
- O número de iterações (tempo de resposta) tende a ser proporcional a $\log_2(n)$:
 - n é a capacidade do vetor;
- Portanto, para $n=1000$, teremos cerca de 10 iterações:
 - $\log_2(1000) = \sim 10$


Busca Binária: Exemplo 2 (1/11)

2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11

- Chave = 6

Busca Binária: Exemplo 2 (2/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 1ª iteração:
 - Início = 0
 - fim = 11

Busca Binária: Exemplo 2 (3/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 1ª iteração:
 - início = 0
 - fim = 11
 - meio = 5

Busca Binária: Exemplo 2 (4/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 1ª iteração:
 - início = 0
 - fim = 11
 - meio = 5 → chave está antes do *meio*!

Busca Binária: Exemplo 2 (5/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 2ª iteração:
 - início = 0
 - fim = 4

Busca Binária: Exemplo 2 (6/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 2ª iteração:
 - início = 0
 - fim = 4
 - meio = 2

Busca Binária: Exemplo 2 (7/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 2ª iteração:
 - início = 0
 - fim = 4
 - meio = 2 → chave está depois do *meio*!

Busca Binária: Exemplo 2 (8/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 3ª iteração:
 - início = 3
 - fim = 4

Busca Binária: Exemplo 2 (9/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 3ª iteração:
 - início = 3
 - fim = 4
 - meio = 3

Busca Binária: Exemplo 2 (10/11)


2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 3ª iteração:
 - início = 3
 - fim = 4
 - meio = 3 → chave está antes do *meio*!

Busca Binária: Exemplo 2 (11/11)

2	4	5	8	11	17	32	44	60	81	90	99
0	1	2	3	4	5	6	7	8	9	10	11



- Chave = 6
- 4ª iteração:
 - início = 3
 - fim = 2 → início > fim (sem espaço de busca!)

Considerações

- Quando a chave não se encontra no vetor, chega-se sempre à condição *inicio > fim*;
 - Não há mais espaço de busca;

Considerações

- Quando a chave não se encontra no vetor, chega-se sempre à condição *inicio > fim*;
 - Não há mais espaço de busca;
- Portanto, os critérios de parada da *busca binária* são:
 - Chave encontrada no *meio*;
 - Chave não encontrada (*inicio > fim*).

Busca Binária: *main()*

```
// programa principal
```

```
int v[] = {2, 4, 5, 8, 11, 17, 32, 44, 60, 81};  
int k = 10, chave;  
printf("Digite o valor a ser buscado: ");  
scanf("%d", &chave);  
int indice = busca_binaria(v, k, chave);  
if(indice == -1)  
    printf("Não encontrado!\n");  
else  
    printf("Encontrado em %d.", indice);
```

Busca Binária: Função Iterativa

```
int busca_binaria (int v[], int n, int chave){
    int ini = 0, fim = n-1, meio;
    do{
        meio = ( ini + fim ) / 2;
        if(v[meio] == chave)
            return meio; // Encontrado em meio!
        if(v[meio] < chave)
            ini = meio + 1;
        else
            fim = meio - 1;
    }while(ini <= fim);
    return -1; // Não encontrado!
}
```

Busca Binária: Função Recursiva

```
int bbin_rec (int v[], int ini, int fim, int chave){  
    if(ini > fim)  
        return -1; // Não encontrado!  
  
}
```


Busca Binária: Função Recursiva

```
int bbin_rec (int v[], int ini, int fim, int chave) {  
    if (ini > fim)  
        return -1; // Não encontrado!  
  
    meio = ( ini + fim ) / 2;  
  
}
```

Busca Binária: Função Recursiva

```
int bbin_rec (int v[], int ini, int fim, int chave) {  
    if (ini > fim)  
        return -1; // Não encontrado!  
  
    meio = ( ini + fim ) / 2;  
  
    if (v[meio] == chave)  
        return meio; // Encontrado em meio!  
  
}
```

Busca Binária: Função Recursiva

```
int bbin_rec (int v[], int ini, int fim, int chave) {  
    if (ini > fim)  
        return -1; // Não encontrado!  
  
    meio = ( ini + fim ) / 2;  
  
    if (v[meio] == chave)  
        return meio; // Encontrado em meio!  
  
    if (v[meio] < chave)  
        return bbin_rec (v, meio + 1, fim, chave);  
  
}
```

Busca Binária: Função Recursiva

```
int bbin_rec (int v[], int ini, int fim, int chave) {  
    if (ini > fim)  
        return -1; // Não encontrado!  
  
    meio = ( ini + fim ) / 2;  
  
    if (v[meio] == chave)  
        return meio; // Encontrado em meio!  
  
    if (v[meio] < chave)  
        return bbin_rec (v, meio + 1, fim, chave);  
  
    // if (v[meio] > chave)  
        return bbin_rec (v, ini, meio - 1, chave);  
}
```


Busca Binária: Função Recursiva

- No programa principal, trocar a linha:

```
int indice = busca_binaria(v, k, chave);
```

- por:

```
int indice = bbin_rec(v, 0, k-1, chave);
```



Exercícios

1. Implemente as funções de busca apresentadas em aula;
2. Modifique a função para *busca binária*, de modo que ela imprima na tela um contador a cada iteração (ou recursão);
 - Analise a quantidade de iterações com um vetor grande (pelo menos 1000 valores);
 - Gere valores aleatórios e ordene o vetor antes da busca.