MC102 – Aula24
Revisão

Instituto de Computação – Unicamp

14 de junho de 2012

Revisão

O que cairá na prova?

- Ordenação;
- Registros;
- Busca;
- Arquivos Texto; e
- Recursão.

Ordenação

- **Selection-Sort**:
  - Pegue o menor elemento, coloque na primeira 0;
  - Pegue o segundo menor elemento, coloque na primeira 1;
  - E assim sucessivamente $\cdots$
- **Bubble-Sort**:
  - Compare vet[0] com vet[1] e troque-os se vet[0] $>$ vet[1].
  - Compare vet[1] com vet[2] e troque-os se vet[1] $>$ vet[2].
  - $\cdots$
  - Compare vet[tam-2] com vet[tam-1] e troque-os se vet[tam-2] $>$ vet[tam-1].
- **Insertion-Sort**:
  - A cada passo, uma porção de 0 até $i-1$ do vetor já está ordenada.
  - Devemos inserir o item da posição i na posição correta para deixar o vetor ordenado até a posição i.
  - No passo seguinte consideramos que o vetor está ordenado até i.

Ordenação

- **Selection-Sort**:
  - Pegue o menor elemento, coloque na primeira 0;
  - Pegue o segundo menor elemento, coloque na primeira 1;
  - E assim sucessivamente $\cdots$
- **Bubble-Sort**:
  - Compare vet[0] com vet[1] e troque-os se vet[0] $>$ vet[1].
  - Compare vet[1] com vet[2] e troque-os se vet[1] $>$ vet[2].
  - $\cdots$
  - Compare vet[tam-2] com vet[tam-1] e troque-os se vet[tam-2] $>$ vet[tam-1].
- **Insertion-Sort**:
  - A cada passo, uma porção de 0 até $i-1$ do vetor já está ordenada.
  - Devemos inserir o item da posição i na posição correta para deixar o vetor ordenado até a posição i.
  - No passo seguinte consideramos que o vetor está ordenado até i.

Ordenação

- **Selection-Sort**:
  - Pegue o menor elemento, coloque na primeira 0;
  - Pegue o segundo menor elemento, coloque na primeira 1;
  - E assim sucessivamente $\cdots$
- **Bubble-Sort**:
  - Compare vet[0] com vet[1] e troque-os se vet[0] $>$ vet[1].
  - Compare vet[1] com vet[2] e troque-os se vet[1] $>$ vet[2].
  - $\cdots$
  - Compare vet[tam-2] com vet[tam-1] e troque-os se vet[tam-2] $>$ vet[tam-1].
- **Insertion-Sort**:
  - A cada passo, uma porção de 0 até $i-1$ do vetor já está ordenada.
  - Devemos inserir o item da posição i na posição correta para deixar o vetor ordenado até a posição i.
  - No passo seguinte consideramos que o vetor está ordenado até i.

Ordenação

Quick-Sort

- Devemos ordenar um vetor de uma posição *ini* até *fim*.
- **Dividir:**
  - Escolha um elemento do vetor especial chamado **pivô**.
  - Particione o vetor em uma posição i tal que todos elementos de *ini* até $i-1$ são menores ou iguais do que o **pivô**, e todos elementos de i até *fim* são maiores ou iguais ao **pivô**.
- Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores (um de *ini* até $i-1$ e o outro de i até *fim*).
- **Conquistar:** Nada a fazer já que o vetor estará ordenado devido a como foi feito a fase de **divisão**.

Ordenação

Quick-Sort

- Devemos ordenar um vetor de uma posição *ini* até *fim*.
- **Dividir:**
  - Escolha um elemento do vetor especial chamado **pivô**.
  - Particione o vetor em uma posição i tal que todos elementos de *ini* até $i-1$ são menores ou iguais do que o **pivô**, e todos elementos de i até *fim* são maiores ou iguais ao **pivô**.
- Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores (um de *ini* até $i-1$ e o outro de i até *fim*).
- **Conquistar:** Nada a fazer já que o vetor estará ordenado devido a como foi feito a fase de **divisão**.

(Instituto de Computação – Unicamp) MC-102 14 de junho de 2012 4 / 19

Ordenação

Quick-Sort

- Devemos ordenar um vetor de uma posição *ini* até *fim*.
- **Dividir:**
  - Escolha um elemento do vetor especial chamado **pivô**.
  - Particione o vetor em uma posição i tal que todos elementos de *ini* até $i-1$ são menores ou iguais do que o **pivô**, e todos elementos de i até *fim* são maiores ou iguais ao **pivô**.
- Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores (um de *ini* até $i-1$ e o outro de i até *fim*).
- **Conquistar:** Nada a fazer já que o vetor estará ordenado devido a como foi feito a fase de **divisão**.

(Instituto de Computação – Unicamp) MC-102 14 de junho de 2012 4 / 19

Registros e Tipos Enumerados

Registros:

```
struct Aluno{
    String nome;
    int RA;
};
struct Aluno aluno;
```

Enumerados:

```
enum meses {jan, fev, mar, abr, mai, jun, jul, ago,
   set,  out, nov, dec};
enum meses a;
a = jan;
```

Typedef:

```
typedef struct Aluno Aluno;
Aluno turma[40];
```


Registros e Tipos Enumerados

Registros:

```
struct Aluno{
    String nome;
    int RA;
};
struct Aluno aluno;
```

Enumerados:

```
enum meses {jan, fev, mar, abr, mai, jun, jul, ago,
   set,  out, nov, dec};
enum meses a;
a = jan;
```

Typedef:

```
typedef struct Aluno Aluno;
Aluno turma[40];
```

(Instituto de Computação – Unicamp) MC-102 14 de junho de 2012 5 / 19

Registros e Tipos Enumerados

Registros:

```
struct Aluno{
    String nome;
    int RA;
};
struct Aluno aluno;
```

Enumerados:

```
enum meses {jan, fev, mar, abr, mai, jun, jul, ago,
   set,  out, nov, dec};
enum meses a;
a = jan;
```

Typedef:

```
typedef struct Aluno Aluno;
Aluno turma[40];
```

(Instituto de Computação – Unicamp) MC-102 14 de junho de 2012 5 / 19

Busca

- **Busca Sequencial**
  - Percorrer o vetor até encontrar o elemento;
  - Se encontrar, retorne a posição, caso contrário retorna -1;
- **Busca Binária**
  - Assume que o vetor estar ordenado;
  - Verifica se a posição do meio é a chave de busca;
  - Caso seja igual devolva a posição;
  - Caso o valor da posição seja maior, repita o processo para a metade do vetor indo até a posição do meio;
  - Caso o valor da posição seja menor, repita o processo para a metade do vetor a partir da posição do meio;

Busca

- **Busca Sequencial**
  - Percorrer o vetor até encontrar o elemento;
  - Se encontrar, retorne a posição, caso contrário retorna -1;
- **Busca Binária**
  - Assume que o vetor estar ordenado;
  - Verifica se a posição do meio é a chave de busca;
  - Caso seja igual devolva a posição;
  - Caso o valor da posição seja maior, repita o processo para a metade do vetor indo até a posição do meio;
  - Caso o valor da posição seja menor, repita o processo para a metade do vetor a partir da posição do meio;

Arquivos Texto

- **Leitura do Arquivo**

```
FILE *f = fopen ("teste.txt", "r");
fscanf(f, "%c", &aux);
fclose(f);
```

- **Escrita no do Arquivo**

```
FILE *fr = fopen ("teste.txt", "r");
FILE *fw = fopen ("saida.txt", "w");
while (fscanf(fr, "%c", &c) != EOF)
    fprintf(fw,"%c", c);
fclose(fr);
fclose(fw);
```


Arquivos Texto

- **Leitura do Arquivo**

```
FILE *f = fopen ("teste.txt", "r");
fscanf(f, "%c", &aux);
fclose(f);
```

- **Escrita no do Arquivo**

```
FILE *fr = fopen ("teste.txt", "r");
FILE *fw = fopen ("saida.txt", "w");
while (fscanf(fr, "%c", &c) != EOF)
    fprintf(fw,"%c", c);
fclose(fr);
fclose(fw);
```


Recursão

Recursividade:

- A idéia é que a solução de um problema pode ser expressa da seguinte forma:
  - Definimos a solução para os casos básicos;
  - Definimos como resolver o problema geral utilizando soluções do mesmo problema só que para casos menores.

Problema:

- Como resolver a * b (inteiros) usando adição?

```
int multiplica(int a, int b){
    int i, res = 0;
    for(i = 1; i<= b; i++)
        res = res + a;
    return res;
}
```

- Como resolver $a * b$ (inteiros) usando adição de forma recursiva?

Recursão

Problema:

- Como resolver a * b (inteiros) usando adição?

```
int multiplica(int a, int b){
    int i, res = 0;
    for(i = 1; i<= b; i++)
        res = res + a;
    return res;
}
```

- Como resolver $a * b$ (inteiros) usando adição de forma recursiva?

Recursão

Problema:

- Como resolver a * b (inteiros) usando adição?

```
int multiplica(int a, int b){
    int i, res = 0;
    for(i = 1; i<= b; i++)
        res = res + a;
    return res;
}
```

- Como resolver $a * b$ (inteiros) usando adição de forma recursiva?

Recursão

- **Vamos definir a solução para os casos básicos:**
  - Se a ou b forem 0 = 0
- Para o caso geral:
  - a = a + a*(b-1) para b > 1

- **Vamos definir a solução para os casos básicos:**
  - Se a ou b forem 0 = 0
- Para o caso geral:
  - a = a + a*(b-1) para b > 1

Resolução:

```
int multiplica(int a, int b){
    if(a ==0 || b == 0)
        return 0;
    else
        return a + multiplica(a, b-1);
}
```


Problema:

- Calcular o fatorial de n de forma recursiva?
- **Casos básicos, como seriam?**
  - Se n = 1 ou 0, então fatorial é 1.
- Para o caso geral:
  - n! = n * (n-1)!, para n > 1

Recursão

Problema:

- Calcular o fatorial de n de forma recursiva?
- **Casos básicos, como seriam?**
  - Se n = 1 ou 0, então fatorial é 1.
- Para o caso geral:
  - n! = n * (n-1)!, para n > 1

(Instituto de Computação – Unicamp) MC-102 14 de junho de 2012 12 / 19

Problema:

- Calcular o fatorial de n de forma recursiva?
- **Casos básicos, como seriam?**
  - Se n = 1 ou 0, então fatorial é 1.
- Para o caso geral:
  - n! = n * (n-1)!, para n > 1

Resolução:

```
int fatorial(int n){
    if(n == 1 || n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```


Problema:

- Calcular *n*-ésimo elemento da sequencia de Fibonacci de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então 0.
  - Se n = 1, então 1.
- Para o caso geral:
  - F(n) = F(n-1) + F(n-2)

Problema:

- Calcular *n*-ésimo elemento da sequencia de Fibonacci de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então 0.
  - Se n = 1, então 1.
- Para o caso geral:
  - F(n) = F(n-1) + F(n-2)

Problema:

- Calcular *n*-ésimo elemento da sequencia de Fibonacci de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então 0.
  - Se n = 1, então 1.
- Para o caso geral:
  - F(n) = F(n-1) + F(n-2)

Resolução:

```
int F(int n){
    if(n == 0) return 0;
    if(n == 1)
        return 1;
    else return F(n-1) + F(n-2);
}
```


Problema:

- Calcular *n*-ésimo elemento da sequencia de Fibonacci de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então 0.
  - Se n = 1, então 1.
- Para o caso geral:
  - F(n) = F(n-1) + F(n-2)

Problema:

- Busca Binária de forma recursiva em um vetor de tamanho n:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna a posição se for igual a chave, caso contrário -1.
- Para o caso geral:
  - Se chave é menor que a posição do meio, repete para o vetor de **ini até meio-1**
  - Se chave é maior que a posição do meio, repete para o vetor de **meio+1 até fim**

Problema:

- Busca Binária de forma recursiva em um vetor de tamanho n:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna a posição se for igual a chave, caso contrário -1.
- Para o caso geral:
  - Se chave é menor que a posição do meio, repete para o vetor de **ini até meio-1**
  - Se chave é maior que a posição do meio, repete para o vetor de **meio+1 até fim**

Recursão

Problema:

- Busca Binária de forma recursiva em um vetor de tamanho n:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna a posição se for igual a chave, caso contrário -1.
- Para o caso geral:
  - Se chave é menor que a posição do meio, repete para o vetor de **ini até meio-1**
  - Se chave é maior que a posição do meio, repete para o vetor de **meio+1 até fim**

Recursão

Problema:

- Busca Binária de forma recursiva em um vetor de tamanho n:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna a posição se for igual a chave, caso contrário -1.
- Para o caso geral:
  - Se chave é menor que a posição do meio, repete para o vetor de **ini até meio-1**
  - Se chave é maior que a posição do meio, repete para o vetor de **meio+1 até fim**

Recursão

Resolução:

```
int busca(int vet[], int ini, int fim, int ch){
    int meio;
    if(fim < ini)
        return -1;
    meio = (ini+fim)/2
    if(vet[meio] == ch)
        return meio;
    else {
        if(vet[meio] > ch)
            return busca(vet, ini, meio-1, ch);
        else return busca(vet, meio+1, fim, ch);
    }
}
```

Recursão

Problema:

- Calcular a soma dos elementos de um vetor de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna o valor da posição (que já seria a soma de 1 elemento).
- Para o caso geral:
  - soma = (soma(ini até meio) + soma(meio+1 até fim))

Recursão

Problema:

- Calcular a soma dos elementos de um vetor de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna o valor da posição (que já seria a soma de 1 elemento).
- Para o caso geral:
  - soma = (soma(ini até meio) + soma(meio+1 até fim))

Recursão

Problema:

- Calcular a soma dos elementos de um vetor de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna o valor da posição (que já seria a soma de 1 elemento).
- Para o caso geral:
  - soma = (soma(ini até meio) + soma(meio+1 até fim))

Recursão

Problema:

- Calcular a soma dos elementos de um vetor de forma recursiva:
- **Casos básicos, como seriam?**
  - Se n = 0, então -1.
  - Se n = 1, retorna o valor da posição (que já seria a soma de 1 elemento).
- Para o caso geral:
  - soma = (soma(ini até meio) + soma(meio+1 até fim))

Resolução:

```
int soma(int vet[], int ini, int fim){
    int meio;
    if(fim < ini)
        return -1;
    meio = (ini+fim)/2
    if(ini == fim)
        return vet[ini];
    else {
            return soma(vet, ini, meio) +
                soma(vet, meio+1, fim);
    }
}
```