Algoritimos de programação II

Professor: Marco Aurélio Stefanes

Lista 01

Aluno:
Augusto Cesar de Aquino Ribas
Análise de Sistemas

1 Aula 01-03 : Exercícios. 1.5 a 1.9

1. (a) Escreva uma função recursiva com a seguinte interface:

```c
int soma_digitos(int n)
```

que receba um número inteiro positivo n e devolva a soma de seus dígitos.

```c
int soma_digitos(int n) {
  int soma;

  if((n/10)==0) {
    soma=n;
  } else {
    soma=n%10+soma_digitosR(n/10);
  }

  return soma;

}
```

(b) Escreva um programa que receba um número inteiro n e imprima a soma de seus dígitos. Use a função do item (a).

```c
#include <stdio.h>

int soma_digitos(int n) {
  int soma;
  if((n/10)==0) {
    soma=n;
  } else {
    soma=n%10+soma_digitosR(n/10);
  }
  return soma;
}

int main(void)
{
  int n;
  scanf("%d",&n);
  printf("Funcao recursiva: Soma dos digitos e %d\n",soma_digitos(n));
  return 0;
}
```

2. A **seqüência de Fibonacci** é uma seqüência de números inteiros positivos dada pela seguinte fórmula:

$$\begin{cases} F_1 & = & 1 \\ F_2 & = & 1 \\ F_i & = & F_{i-1} + F_{i-2}, \text{ para } i \geq 3 \end{cases}$$

(a) Escreva uma função recursiva com a seguinte interface:

```
int Fib(int i)
```

que receba um número inteiro positivo i e devolva o i-ésimo termo da seqüência de Fibonacci, isto é, F_i.

```
int fib(int i){

  if(i==1) {
    return 1;
  } else {
    if(i==2) {
      return 1;
    } else {
      return fib(i-1)+fib(i-2);
    }
  }
}
```

(b) Escreva um programa que receba um número inteiro $i \geq 1$ e imprima o termo F_i da seqüência de Fibonacci. Use a função do item (a).

```
#include <stdio.h>

int fib(int i){
  if(i==1) {
    return 1;
  } else {
    if(i==2) {
      return 1;
    } else {
      return fib(i-1)+fib(i-2);
    }
  }
}

int main(void)
{
  int i;
  scanf("%d",&i);
  printf("Resultado = %d\n", fib(i));
  return 0;
}
```

3. O **piso** de um número inteiro positivo x é o único inteiro i tal que $i \leq x < i + 1$. O piso de x é denotado por $\lfloor x \rfloor$.

   Segue uma amostra de valores da função $\lfloor \log_2 n \rfloor$:

n	15	16	31	32	63	64	127	128	255	256	511	512
$\lfloor \log_2 n \rfloor$	3	4	4	5	5	6	6	7	7	8	8	9

   (a) Escreva uma função recursiva com a seguinte interface:

```
int piso_log2(int n)
```

   que receba um número inteiro positivo n e devolva $\lfloor \log_2 n \rfloor$.

   (b) Escreva um programa que receba um número inteiro $n \geq 1$ e imprima $\lfloor \log_2 n \rfloor$. Use a função do item (a).

4. Considere o seguinte processo para gerar uma seqüência de números. Comece com um inteiro n. Se n é par, divida por 2. Se n é ímpar, multiplique por 3 e some 1. Repita esse processo com o novo valor de n, terminando quando $n = 1$. Por exemplo, a seqüência de números a seguir é gerada para $n = 22$:

   22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

   É conjecturado que esse processo termina com $n = 1$ para todo inteiro $n > 0$. Para uma entrada n, o **comprimento do ciclo de** n é o número de elementos gerados na seqüência. No exemplo acima, o comprimento do ciclo de 22 é 16.

   (a) Escreva uma função não-recursiva com a seguinte interface:

   ```
   int ciclo(int n)
   ```

   que receba um número inteiro positivo n, mostre a seqüência gerada pelo processo descrito acima na saída e devolva o comprimento do ciclo de n.

   ```
   int ciclo(int n){
     int ciclo=1;
     while(n!=1) {
       if(n%2==0) {
         printf("%d ",n/2);
         n=n/2;
       } else {
         printf("%d ",n*3+1);
         n=n*3+1;
       }
       ciclo++;
     }
     return ciclo;
   }
   ```

(b) Escreva uma versão recursiva da função do item (a) com a seguinte interface:

```c
int cicloR(int n)
```

que receba um número inteiro positivo n, mostre a seqüência gerada pelo processo descrito acima na saída e devolva o comprimento do ciclo de n.

```c
int cicloR(int n){
  if(n%2==0) {
    printf("%d ",n/2);
    n=n/2;
  } else {
    printf("%d ",n*3+1);
    n= n*3+1;
  }
  if(n==1) {
    return 2;
  } else {
    return 1+cicloR(n);
  }
}
```

(c) Escreva um programa que receba um número inteiro $n \geq 1$ e determine a seqüência gerada por esse processo e também o comprimento do ciclo de n. Use as funções em (a) e (b) para testar.

```c
#include <stdio.h>

int ciclo(int n){
  int ciclo=1;
  while(n!=1) {
    if(n%2==0) {
      printf("%d ",n/2);
      n=n/2;
    } else {
      printf("%d ",n*3+1);
      n=n*3+1;
    }
    ciclo++;
  }
  return ciclo;
}
int cicloR(int n){
  if(n%2==0) {
    printf("%d ",n/2);
    n=n/2;
  } else {
    printf("%d ",n*3+1);
    n= n*3+1;
  }
  if(n==1) {
    return 2;
  } else {
    return 1+cicloR(n);
  }
}

int main(void)
{
  int x, saida;
  scanf("%d",&x);
  saida=ciclo(x);
  printf("\niterativo %d\n",saida);
  saida=cicloR(x);
  printf("\nrecursivo %d\n",saida);
  return 0;
}
```

5. Podemos calcular a potência x^n de uma maneira mais eficiente. Observe primeiro que se n é uma potência de 2 então x^n pode ser computada usando seqüências de quadrados. Por exemplo, x^4 é o quadrado de x^2 e assim x^4 pode ser computado usando somente duas multiplicações ao invés de três. Esta técnica pode ser usada mesmo quando n não é uma potência de 2, usando a seguinte fórmula:

$$x^n = \begin{cases} 1\,, & \text{se } n = 0, \\ (x^{n/2})^2\,, & \text{se } n \text{ é par,} \\ x \cdot x^{n-1}\,, & \text{se } n \text{ é ímpar.} \end{cases}$$

   (a) Escreva uma função com interface

```c
int potencia(int x, int n)
```

   que receba dois números inteiros x e n e calcule e devolva x^n usando a fórmula acima.

```c
int potencia(int x, int n) {
  int num;
  if(n==0) {
    printf("N e %d \n",n);
    /*Se n e zero*/
    return 1;
  } else {
    if( (n%2) == 0 ) {
      printf("N e %d \n",n);
    /*Se n e par*/
      num=potencia(x,n/2);
      return num*num;
    } else {
      printf("N e %d \n",n);
    /*Se n e impar*/
      return x * potencia(x,n-1);
    }
  }
}
```

(b) Escreva um programa que receba dois números inteiros a e b e imprima o valor de a^b.

```c
#include <stdio.h>

int potencia(int x, int n) {
  int num;
  if(n==0) {
    printf("N e %d \n",n);
    /*Se n e zero*/
    return 1;
  } else {
    if( (n%2) == 0 ) {
      printf("N e %d \n",n);
    /*Se n e par*/
      num=potencia(x,n/2);
      return num*num;
    } else {
      printf("N e %d \n",n);
    /*Se n e impar*/
      return x * potencia(x,n-1);
    }
  }
}

int main(void) {
  int num, pot;
  scanf("%d %d",&num, &pot);
  printf("Resposta: %d\n",potencia(num,pot));
  return 0;
}
```

2 Aula 05: Exercícios 2.6 a 2.10

1. É verdade que $2^{n+1} = O(2^n)$? E é verdade que $2^{2 \cdot n} = O(2^n)$?

2. Suponha que você tenha algoritmos com os cinco tempos de execução listados abaixo. Quão mais lento cada um dos algoritmos fica quando você (i) duplica o tamanho da entrada, ou (ii) incrementa em uma unidade o tamanho da entrada?

   (a) n^2
   (b) n^3
   (c) $100 \cdot n^2$
   (d) $n \cdot \log_2(n)$
   (e) 2^n

3. Suponha que você tenha algoritmos com os seis tempos de execução listados abaixo. Su- ponha que você tenha um computador capaz de executar 1010 operações por segundo e você precisa computar um resultado em no máximo uma hora de computação. Para cada um dos algoritmos, qual é o maior tamanho da entrada n para o qual você poderia receber um resultado em uma hora?

   (a) n^2
   (b) n^3
   (c) $100 \cdot n^2$
   (d) $n \cdot \log_2(n)$
   (e) 2^n
   (f) 2^{2^n}

4. Rearranje a seguinte lista de funções em ordem crescente de taxa de crescimento. Isto é, se a função $g(n)$ sucede imediatamente a função $f(n)$ na sua lista, então é verdade que $f(n) = O(g(n))$.

$$f1(n) = n^{2.5}$$
$$f2(n) = \sqrt{2 \cdot n}$$
$$f3(n) = n + 10$$
$$f4(n) = 10^n$$
$$f5(n) = 100^n$$
$$f6(n) = n^2 \cdot \log_2(n)$$

5. Considere o problema de computar o valor de um polinômio em um ponto. Dados n coeficientes $a_0, a_1, ..., a_{n-1}$ e um número real x, queremos computar $\Sigma_{i=0}^{n-1} a_i \cdot x^i$.

   (a) Escreva um programa simples com tempo de execução de pior caso $O(n^2)$ para solucionar este problema.

(b) Escreva um programa com tempo de execução de pior caso $O(n)$ para solucionar este problema usando o método chamado de regra de Horner para reescrever o polinômio:

6. Seja $A[0...n-1]$ um vetor de n números inteiros distintos dois a dois. Se $i < j$ e $A[i] > A[j]$ então o par (i, j) é chamado uma inversão de A.

   (a) Liste as cinco inversões do vetor $A = 2, 3, 8, 6, 1$.

   (b) Qual vetor com elementos no conjunto $1, 2, ..., n$ tem a maior quantidade de in- versões? Quantas são?

   (c) Escreva um programa que determine o número de inversões em qualquer permuta- ção de n elementos em tempo de execução de pior caso $O(n \cdot \log_2(n))$.