Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2013

Soma

Soma de dois números inteiros não negativos, x e y.

```
int soma(int x, int y) {
  if (y == 0)
    return x;
  else
    return soma(x, y - 1) + 1;
}
```

Multiplicação

Multiplicação de dois números inteiros não negativos, x e y.

```
int mult(int x, int y) {
  if (y == 0)
    return 0;
  else
    if (y == 1)
      return x;
    else
      return mult(x, y - 1) + x;
}
```

Soma de valores pares

Soma de todos os valores pares positivos menores ou iguais a um valor inteiro n.

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

Produtório

Cálculo do produtório $\prod_{i=m}^{n} i = m \times (m+1) \times (m+2) \cdots n$, tal que $m \leq n$.

```
int produtorio(int m, int n)
  if (m == n)
    return m;
  return m * produtorio(m + 1, n);
}
```

Potência

Cálculo de potência x^n, para $n > 0$.

$$x^n = \begin{cases} 1 & \text{se } n = 0 \\ xx^{n-1} & \text{caso contrário} \end{cases}$$

```
float potencia(float x, int n) {
  if (n == 0)
    return 1;
  else
    return x * potencia(x, n - 1);
}
```

Potência

Neste caso, a solução iterativa é mais eficiente.

```
float potencia(float x, int n) {
  float p = 1, i;
  for (i = 1; i <= n; i++)
    p = p * x;
  return p;
}
```

- O laço de repetição é executado n vezes.
- Na solução recursiva, n chamadas são realizadas, entretanto, há o custo adicional para criação/remoção de variáveis locais na pilha.

Potência

Podemos definir x^n de uma forma diferente:

- Caso básico:
  - Se $n = 0$, então $x^n = 1$.
- Caso geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note que definimos a solução do caso mais complexo em termos de casos mais simples. Usando esta definição podemos implementar uma função iterativa ou recursiva, ambas mais eficientes que as versões anteriores.

Potência

```
float potencia(float x, int n) {
  float aux;

  if (n == 0)
    return 1;
  else
    if (n % 2 == 0) { /* se n é par */
      aux = potencia(x, n/2);
      return aux * aux;
    } else { /* se n é impar */
      aux = potencia(x, (n - 1)/2);
      return x * aux * aux;
    }
}
```

Potência

- Na nova versão do algoritmo, a cada chamada recursiva, o valor de n é dividido por 2. Ou seja, a cada chamada recursiva, o valor de n decai para pelo menos a metade.
- Usando divisões inteiras, faremos no máximo $\lceil \log_2 n \rceil + 1$ chamadas recursivas.
- Por outro lado, o algoritmo iterativo original executa o laço n vezes.

Máximo Divisor Comum

Máximo divisor comum entre x e y, ambos não negativos.

```
int mdc(int x, int y) {
  if (y == 0)
    return x;
  else
    return mdc(y, x % y);
}
```

Maior elemento de um vetor

Maior elemento de um vetor v de $n > 0$ números inteiros.

```
int max(int v[], int n) {
  int x;

  if (n == 1)
    return v[0];
  else {
    x = max(v, n - 1);
    if (x > v[n - 1])
      return x;
    else
      return v[n - 1];
  }
}
```

Soma dos dígitos de um inteiro

Soma dos dígitos de um número inteiro positivo.

```
/* Versao 1 */
int soma_digitos(int n) {
  if (n == 0)
    return 0;
  else
    return soma_digitos(n / 10) + (n % 10);
}

/* Versao 2 */
int soma_digitos(int n) {
  if (n < 10)
    return n;
  else
    return soma_digitos(n / 10) + (n % 10);
}
```

Número de caracteres de uma string

Calcula número de caracteres de uma string.

```
int strlen(char *s) {
  if (*s == '\0')
    return 0;
  else
    return strlen(s + 1) + 1;
}
```

Comparação de strings

Compara duas strings e retorna 0 se as strings são iguais, um valor negativo se a primeira string é lexicograficamente menor que a segunda ou um valor positivo se a primeira string é lexicograficamente maior que a segunda.

```
int strcmp(char *s, char *t) {
  if ((*s == '\0') || (*s != *t))
    return *s - *t;
  else
    return strcmp(s + 1, t + 1);
}
```

Busca de um caractere em uma string

Função que busca um caractere em uma string e retorna o ponteiro para ele (caso encontre).

```
char *strchr(char *s, char c) {
  if (*s == c)
    return s;
  else
    if (*s == '\0')
      return NULL;
    else
      return strchr(s + 1, c);
}
```

Cópia de strings

Função que retorna uma cópia da string `t` na string `s`.

```
void strcpy(char *s, char *t) {
  *s = *t;
  if (*s)
    strcpy(s + 1, t + 1);
}
```

Palíndromo

Verifica se uma string é um palindromo.

```
int palindromo(char *s, int n) {
  if (n <= 1)
    return 1;
  if (s[0] == s[n - 1])
    return palindromo(s + 1, n - 2);
  else
    return 0;
}
```

Impressão de uma string em ordem inversa

Imprime string em ordem inversa.

```
void imprime_reversa(char *s) {
  if (*s) {
    imprime_reversa(s + 1);
    printf("%c", *s);
  }
}
```

Impressão

O que será impresso pela chamada `imprimir(5)`?

```
void imprimir(int i) {
  int j;
  if (i > 0) {
    imprimir(i - 1);
    for (j = 1; j <= i; j++)
       printf("*");
    printf("\n");
  }
}
```

```
*
**
***
****
*****
```

Torre de Hanoi

- Considere n discos de diâmetros diferentes colocados em um pino A.
- O problema da Torre de Hanoi consiste em transferir os n discos do pino A para o pino C, usando um pino B como auxiliar.
- Entretanto, deve-se respeitar algumas regras:
  - Apenas o disco do topo de um pino pode ser movido.
  - Nunca um disco de diâmetro maior pode ficar sobre um disco de diâmetro menor.
- O problema foi descrito pela primeira vez no ocidente em 1883 pelo matemático francês Édouard Lucas, baseado numa lenda hindu, onde Brahma havia ordenado que os monges do templo de Kashi Vishwanath movessem uma pilha de 64 discos de ouro, segundo as regras previamentes descritas.
- Quando todos os discos tivessem sido movidos, o mundo acabaria.

Torre de Hanoi

- Vamos usar indução para obter um algoritmo para este problema.

Teorema

É possível resolver o problema da Torre de Hanoi com n discos.

Torre de Hanoi

Prova.

- Base da Indução: $n = 1$. Neste caso, temos apenas um disco. Basta mover este disco do pino A para o pino C.
- Hipótese de Indução: Sabemos como resolver o problema quando há $n - 1$ discos.
- Passo de Indução: Devemos resolver o problema para n discos assumindo que sabemos resolver o problema com $n - 1$ discos.
- Por hipótese de indução, sabemos mover os $n - 1$ primeiros discos do pino A para o pino B usando o pino C como auxiliar.
- Depois de movermos estes $n - 1$ discos, movemos o maior disco (que continua no pino A) para o pino C.
- Novamente, pela hipótese de indução, sabemos mover os $n - 1$ discos do pino B para o pino C usando o pino A como auxiliar.
- Com isso, temos uma solução para o caso em que há n discos.

□

Torre de Hanoi

Como solucionar o problema de forma recursiva:

1. Se $n = 1$ então mova o único disco de A para C.
2. Caso contrário ($n > 1$) desloque de forma recursiva os $n - 1$ primeiros discos de A para B, usando C como auxiliar.
3. Mova o último disco de A para C.
4. Mova, de forma recursiva, os $n - 1$ discos de B para C, usando A como auxiliar.

Torre de Hanoi

```
#include<stdio.h>

void hanoi(int n, char inicial, char final, char auxiliar) {
  if (n == 1)
    printf("Mova o disco %d do pino %c para o pino %c\n", n, inicial, final);
  else {
    hanoi(n - 1, inicial, auxiliar, final);
    printf("Mova o disco %d do pino %c para o pino %c\n", n, inicial, final);
    hanoi(n - 1, auxiliar, final, inicial);
  }
}

int main() {
  int n;

  printf("Entre com o numero de discos: ");
  scanf("%d", &n);

  hanoi(n, 'A', 'C', 'B');

  return 0;
}
```

Torre de Hanoi

- Solução para 4 discos:

```
Mova o disco 1 do pino A para o pino B
Mova o disco 2 do pino A para o pino C
Mova o disco 1 do pino B para o pino C
Mova o disco 3 do pino A para o pino B
Mova o disco 1 do pino C para o pino A
Mova o disco 2 do pino C para o pino B
Mova o disco 1 do pino A para o pino B
Mova o disco 4 do pino A para o pino C
Mova o disco 1 do pino B para o pino C
Mova o disco 2 do pino B para o pino A
Mova o disco 1 do pino C para o pino A
Mova o disco 3 do pino B para o pino C
Mova o disco 1 do pino A para o pino B
Mova o disco 2 do pino A para o pino C
Mova o disco 1 do pino B para o pino C
```

Torre de Hanoi

- Seja $T(n)$ o número de movimentos necessários para mover uma pilha de n discos.
- Claramente temos que:
  - $T(1) = 1$
  - $T(n) = 2T(n-1) + 1$
- O que nos permite deduzir que:
  - $T(2) = 2T(1) + 1 = 3$
  - $T(3) = 2T(2) + 1 = 7$
  - $T(4) = 2T(3) + 1 = 15$
  - $T(5) = 2T(4) + 1 = 31$
  - $\ldots$
  - $T(n) = 2^n - 1$
- No caso de 64 discos são necessários 18.446.744.073.709.551.615 movimentos ou, aproximadamente, 585 bilhões de anos, se cada movimento puder ser feito em um segundo.