MAC–IME–USP

Carlos Eduardo Ferreira
Sala 108C Tel.: 3091 6135
e-mail cef@ime.usp.br
Monitores: Alexandre Freire e João Miranda

MAC 122 – Princípios de Desenvolvimento de Algoritmos

Segundo semestre de 2009

Lista de Exercícios para estudar para a primeira prova

1 Recursão

"Para fazer uma função recursiva
é preciso ter fé."
Siang Wun Song

1. (a) Faça uma função recursiva MaxMin que calcula o elemento máximo e o elemento mínimo de um vetor com n números inteiros.
   (b) Quantas comparações (em função de n) envolvendo elementos do vetor o seu algoritmo faz?

2. Considere a função abaixo que calcula o n-ésimo termo da série de Fibonacci:

```
int fibonacci(int n)
{
  printf("*");
  if ((n == 1) || (n == 2))
    return (1);
  return (fibonacci(n-1) + fibonacci(n-2));
}
```

   Quantas "*" são impressas no cálculo de `fibonacci(n)`? Prove.

3. Faça uma função recursiva Dígito que recebe um número inteiro n e calcula a soma dos digitos de n. Exemplo: se $n = 132$ então Dígito$(n) = 6$.

4. Considere a função abaixo:

```
double f(double x, double y)
{
  if (x >= y)
     return ((x+y)/2);
  return (f(f(x+2, y-1), f(x+1, y-2));
}
```

Qual é o valor de $f(1, 10)$? Como se poderia calcular $f(a, b)$ de maneira mais simples? Prove.

5. A função de Ackermann é definida da seguinte maneira:

$$A(m,n) := \begin{cases} n+1 & \text{se } m = 0, \\ A(m-1,1) & \text{se } m > 0 \text{ e } n = 0, \\ A(m-1, A(m,n-1)) & \text{se } m, n > 0. \end{cases}$$

   Escreva uma função recursiva que recebe inteiros não negativos m e n e devolve $A(m, n)$.

6. Simule a execução do programa abaixo:

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

int main()
{
  int m = 7;
  printf("Fusc = %d\n", fusc(m));
}
```

7. Considere a seguinte função:

```
void misterio (int A[], int inic, int fim)
{
  int aux;
  while (A[fim] % 2 == 0 && inic < fim)
     fim --;
  while (A[inic] % 2 == 1 && inic < fim)
     inic++;
  if (inic < fim){
    aux = A[inic];
    A[inic] = A[fim];
    A[fim] = aux;
    misterio(A, inic, fim);
  }
}
```

   (a) Simule a função mistério para

	0	1	2	3	4	5	6	7	8
$A =$	8	10	3	6	5	2	9	1	4

   Inicio$= 0$ e Fim $= 8$.

(b) O que faz a função mistério? Quantas comparações envolvendo elementos do vetor A são feitas? Escreva um algoritmo que faz a mesma coisa com um número menor de comparações.

8. Simule a seguinte função recursiva para $n = 6$:

```
int zzz(int n)
{
  int aux;
  if (n <= 2)
    return(1);
  n--;
  aux = zzz(n);
  n--;
  return (aux + zzz(n));
}
```

O que faz a função `zzz`?

9. Escreva uma função recursiva `Tabela` que recebe como parâmetro um inteiro não negativo n e calcula um par de inteiros (x, y), onde x e y são as coordenadas de n na tabela abaixo:

	0	1	2	3	4	...	Y
0	0	2	5	9	14		
1	1	4	8	13			
2	3	7	12				
3	6	11					
4	10						
...							
X							N

10. A função abaixo calcula o maior divisor comum dos inteiros positivos m e n. Escreva uma função recursiva equivalente.

```
int Euclides (int m, int n)
{
  int r=m%n;
  while (r != 0){
     m = n;
     n = r;
     r = m % n;
  }
  return(n);
}
```

2 Listas Ligadas

Nos exercícios abaixo considere que a lista ligada é dada através de um apontador `inicio` para sua primeira célula. A lista ligada é definida através dos seguintes tipos:

```
typedef struct no {
                    int         info;
                    struct no * prox;
                  } celula;
typedef celula * apontador;
```

1. Escreva uma função que recebe uma lista ligada ordenada e remove da lista os elementos repetidos, deixando apenas uma cópia de cada elemento.

2. Escreva funções para as seguintes operações:

   (a) verifica se um dado `x` ocorre numa lista ligada;

   (b) acrescenta um elemento no fim de uma lista ligada;

   (c) imprima os conteúdos de todos os elementos de uma lista ligada.

3. Faça uma função que devolve um apontador para o elemento do meio de uma lista ligada (se o número de elementos da lista for par, devolve o $\frac{n}{2}$-ésimo elemento) **sem contar o número de elementos da lista**.

4. Discuta vantagens e desvantagens de vetores (arrays) em relação a listas ligadas. Dê atenção especial às questões de quantidade de memória, velocidade de inserção, remoção e acesso.

5. Escreva uma função que inverte uma lista ligada dada (o primeiro elemento da nova lista é o último da lista dada, o segundo é o penúltimo da lista dada, e assim por diante. Faça manipulando apenas os apontadores. Exemplo:

6. Escreva uma função para intercalar duas listas ligadas cujas informações estão arranjadas em ordem crescente. Exemplo:

7. Dada uma lista ligada escreva uma função que transforma a lista dada em duas listas ligadas: a primeira contendo os elementos cujos conteúdos são pares e a segunda com os elementos cujos conteúdos são ímpares. Sua função deve manipular somente os apontadores e **não** o conteúdo das células (i.e. não vale ficar copiando os conteúdo de um lado para o outro, só vale alterar os apontadores). Exemplo:

3 Pilhas

1. Sejam os inteiros 1, 2, 3 e 4 que são lidos nesta ordem para serem colocados numa pilha. Considerando-se todas as possíveis seqüências de operações *Empilha* e *Desempilha* decida quais das 24 (4!) permutações possíveis podem ser obtidas como saída da pilha. Por exemplo, a permutação 2, 3, 1, 4 pode ser obtida da seguinte forma: *Empilha* 1, *Empilha* 2, *Desempilha* 2, *Empilha* 3, *Desempilha* 3, *Desempilha* 1, *Empilha* 4, *Desempilha* 4.

2. Na seqüência abaixo considere que uma letra significa "Empilha" e um "*" significa "Desempilha":

   UTA*EC**R**O**

   Qual é a seqüência em que as letras são desempilhadas?

3. Usando a notação do exercício anterior, é possível incluir "*" nas seqüências abaixo para produzir a palavra ALGORITMO como resultado?

- LGAROMOTI
- ALGORITMO
- OMTIROGLA
- OTAIGLROM

4. Seja P o seguinte conjunto de cadeia sobre $\{a, b, c\}$:

$$P = \{c, aca, bcb, abcba, bacab, bbcbb, \ldots\}$$

Uma cadeia deste conjunto pode ser especificada por $\alpha c \alpha^{-1}$, onde α é uma seqüência de letras que só contém $a's$ e $b's$ e α^{-1} é o reverso de α, ou seja, α lido de trás para frente. Dada uma cadeia β, faça um programa que determina se β pertence ou não a P, ou seja, determina se β é da forma $\alpha c \alpha^{-1}$ para alguma cadeia α.

5. Escreva um algoritmo, usando uma *Pilha*, que inverte as letras de cada palavra de um texto terminado por ponto ('.') preservando a ordem das palvras. Por exemplo, dado o texto:

ESTE EXERCÍCIO É MUITO FÁCIL.

a saída deve ser

ETSE OICÍCREXE É OTIUM LICÁF.

6. Simule a execução do algoritmo de conversão para a notação posfixa com a expressão aritmética abaixo:

$$(A + B) * D + E/(F + A * D) + C$$

7. Considerando o algoritmo de conversão para a notação posfixa responda às seguintes perguntas.

   (a) Qual é o tamanho máximo que a pilha pode atingir se a expressão a ser traduzida tiver tamanho n (i.e., o numero total de operandos, operadores, e abre e fecha parêntesis na expressão é n. Pode supor que a expressão está correta.)?

   (b) Qual é a resposta para o item (a) se restringirmos o número de parêntesis na expressão para no máximo 6 (número de pares abre e fecha parêntesis)?

8. Uma outra forma de expressão sem parêntesis que é fácil de ser avaliada é chamada de notação prefixa. Nesta forma de escrever as expressões aritméticas os operadores precedem seus operandos. Por exemplo:

infixa	prefixa
$A * B/C$	$/ * ABC$
$A * (D + C)/B - G$	$-/ * A + DCBG$
$A + B * C - D@E + A * B$	$+ - + A * BC@DE * AB$

Observe que a ordem dos operandos não é alterada passando da notação infixa para a prefixa.

(a) Passe a expressão aritmética do exercício 5 para a notação prefixa.

(b) Escreva um algoritmo que transforma uma expressão na forma infixa para a expressão prefixa correspondente.

Sugestão: percorra a expressão infixa de trás para a frente.

9. Dados inteiros n e k considere um tabuleiro de xadrez $n \times n$ e faça um programa que imprima de quantos possíveis jeitos podemos dispor k bispos neste tabuleiro de forma que eles não se ataquem. Use backtrack para decidir, para cada um dos k bispos onde colocá-lo.

10. Considere um ladrão que enfrenta o seguinte problema. Ele tem à sua disposição n objetos que pode roubar. Cada um deles tem um peso p_i e um certo valor v_i. Para carregá-los ele dispõe de uma mochila de capacidade C. Faça um programa que ajuda o ladrão a encontrar o roubo ótimo, ou seja, o conjunto de objetos de melhor valor possível que caiba na mochila. Use backtrack para decidir se vai levar cada um dos objetos ou não.