Lista de Exercícios de CES-11
CTA - ITA - IEC

Importante: Não vale nota, ou seja, não é preciso entregar!

A. Passagem de parâmetros, escopo de variáveis, recursão

1. Analise o código abaixo. Qual é a sua saída?

```
#include <stdio.h>
#include <conio.h>

int i = 10, j = 20;

void g (int i, int j, int k, int l) {
  printf ("%7d%7d%7d%7d\n\n", i, j, k, l);
}

void f (int *m, int *n, int p, int q) {
  int k, l;
  g (*m, *n, p, q); k = 5; l = 8;
  *m = 500; *n = 600; p = 800; q = 700;
  g (i, j, k, l); g (*m, *n, p, q);
}

void main () {
  int i, j, k, l;
  i = 6; j = 7; k = 8; l = 9; g (i, j, k, l);
  {
        int j, k;
        i = 44; j = 45; k = 46; l = 47;
        g (i, j, k, l); f (&j, &k, i, l);
  }
  g (i, j, k, l);

  getche();
}
```

2. A sequência de *Fibonacci* é definida por: 1, 1, 2, 3, 5, 8, 13, 21, ... Repare que os dois primeiros termos são o número 1, e os demais são a soma dos dois termos predecessores. Escreva um algoritmo iterativo que calcule e imprima o n-ésimo termo dessa sequência. Idem, mas com um algoritmo recursivo **`Fib(n)`**.

3. Simule a pilha de execução para **`Fib(4)`**. Seu computador consegue calcular **`Fib(50)`**? Por quê?

4. Escreva uma função recursiva que recebe um número inteiro n>0 e calcula o valor da série 1 + 1/2! + 1/3! + ... + 1/n!.

5. Simule a pilha de execução do código abaixo. O que ele faz?

```
#include <stdio.h>
#include <conio.h>

int rec (int x[], int n) {
   if (n < 0) return 0;
   if (x[n] > 0) return x[n] + rec (x, n-1);
   else return rec (x, n-1);
}

void main() {
   int v[5] = {-1, 0, 1, -2, 3};
   printf("%d\n", rec(v,4));
   getche();
}
```

6. Escreva uma função recursiva que imprima o elemento de maior valor presente em um vetor de n inteiros. O vetor também deve ser recebido como parâmetro, e seus índices variam de 0 a n-1.

7. Escreva uma função recursiva que recebe dois números inteiros, e retorna o resultado do somatório de todos os números contidos no intervalo aberto delimitado por esses valores.

8. Escreva uma função recursiva que recebe um número inteiro n>0 e imprime as sequências de n a 1 e de 1 a n.

Exemplo para n=10: 10 9 8 7 6 5 4 3 2 1 1 2 3 4 5 6 7 8 9 10

9. Escreva uma função recursiva que recebe um número decimal positivo e retorna seu valor em binário.

a) Elabore uma versão em que o resultado é escrito na tela.

b) Elabore uma versão em que a função retorna um número inteiro, formado apenas por 0's e 1's, que corresponde ao resultado calculado.

10. Escreva uma função que recebe um número inteiro n>0 e gere recursivamente uma sequência com todas as representações de n *bits*, de tal modo que, entre duas sucessivas, haja um único *bit* distinto.

Dica (Gray Code): A solução para n *bits* pode ser obtida do seguinte modo:

a) Gere a solução para n-1 *bits*.

b) Adicione um *bit* 0 mais à esquerda de cada elemento, formando a sequência x.

c) Gere a mesma solução para n-1 *bits*, mas agora em ordem invertida.

d) Adicione um *bit* 1 mais à esquerda de cada elemento, formando a sequência y.

e) Concatene a sequência x com a y.

B. Listas, filas e pilhas

1. Dadas duas **listas lineares**, deseja-se implementar a operação de **concatenação**, isto é, acrescentar a segunda lista no final da primeira. Qual seria a melhor implementação para realizar essa operação: vetor ou lista ligada? Por quê? Descreva como seriam os códigos.

2. Seja uma **lista encadeada com nó-líder** que armazena valores inteiros:

a) Supondo que essa lista já esteja ordenada crescentemente, implemente uma função que insira um novo valor inteiro, mantendo a ordenação da lista.

Dica: Use a função **`inserirElemento`** vista em sala de aula.

b) A partir da função desenvolvida acima, implemente uma função que leia uma sequência qualquer de inteiros e armazene-os de *modo ordenado* em uma lista ligada. Qual é a complexidade de tempo dessa função?

3. Construa uma função que recebe como parâmetros uma **lista** L1 e um inteiro positivo x, e que retira os primeiros x elementos da lista L1, inserindo-os no fim de L1, sem alterar a ordenação inicial desses elementos. *Importante*: É proibido o uso de uma lista auxiliar!

Faça isso nos seguintes casos:

a) Lista implementada como estrutura contígua (vetor);

b) Lista implementada como estrutura encadeada (nós).

Observação: Fique à vontade para usar ou não o nó-líder, mas tenha em conta que, de um modo ou de outro, as operações da lista deverão funcionar corretamente.

4. Considere uma **lista encadeada com nó-líder** contendo apenas valores 0 e 1. Escreva uma função que a receba como parâmetro e retorne as posições inicial e final da maior sequência S de elementos nulos dentro da lista. Exemplo: na lista {0,1,1,0,0,0,1,0}, as posições inicial e final serão respectivamente 4 e 6, pois S = {0,0,0}.

Observação: No caso em que haja mais de uma resposta possível, retorne a primeira sequência.

5. Considere a seguinte representação de uma **lista encadeada**:

```
struct node {int elem; node *prox;};
struct fila {node *fr, *tr;};
fila F;
```

Supondo que haja nó-líder, implemente as operações:

a) **`void inicFila (fila *F)`**: inicializa **`F`**, deixando-a pronta para receber elementos. Como resultado, espera-se que **`F`** esteja com os ponteiros `fr` e `tr` apontando para o nó-líder.

b) **`bool isEmpty (fila *F)`**: retorna *true* se **`F`** estiver vazia, ou *false*, caso contrário.

c) **`void front (fila *F)`**: escreve na tela o primeiro elemento de **`F`**, caso exista.

d) **`void enqueue (fila *F, int x)`**: inclui um elemento no final de **`F`**.

e) **`void dequeue (fila *F)`**: retira o elemento que está no início de **`F`**. *Importante*: É preciso considerar os casos em que haja apenas um ou nenhum elemento em **`F`**.

6. O *problema de Josephus* consiste na escolha de um vencedor, no qual os participantes são eliminados até sobrar apenas um. O grupo inicial, com n pessoas, é organizado em um círculo, onde os participantes são numerados sequencialmente de 1 a n. A partir do número 1, a pessoa consecutiva é eliminada, e o processo recomeça de modo semelhante a partir do próximo participante não eliminado. Tudo termina quando restar apenas uma pessoa, que será a vencedora. O objetivo deste problema é, dado um determinado n>0, descobrir o número J(n) do vencedor. Alguns casos particulares: J(1)=1, J(2)=1, J(3)=3, J(4)=1, J(5)=3, J(6)=5, J(7)=7, etc. Escreva um programa que, a partir de um inteiro n positivo, simule o *problema de Josephus* e calcule J(n). Para isso, deverá implementar uma lista encadeada circular, onde cada nó representa um participante. A eliminação de uma pessoa corresponderá à remoção do seu nó. Ao final do processo, o número do nó restante será a resposta do problema. A partir de diversos valores de n, monte uma tabela com os resultados. Você consegue descobrir qual é a lei de formação de J(n)?

7. Seja um vetor de números inteiros **`int vet[N]`**, onde **`N`** é um inteiro positivo definido. Proponha um algoritmo que inverta as posições do vetor, usando como estrutura auxiliar uma pilha e suas operações básicas: *inicPilha*, *push*, *pop*, *top*, *isEmpty*.

8. Proponha um algoritmo para calcular o valor em binário de um número decimal, usando como estrutura auxiliar uma pilha e suas operações básicas: *inicPilha*, *push*, *pop*, *top*, *isEmpty*.

9. Mostre como duas filas podem simular uma pilha. Analise o tempo das operações *push* e *pop* dessa simulação.

10. Mostre como duas pilhas podem simular uma fila. Analise o tempo das operações *enqueue* e *dequeue* dessa simulação.

11. Um palíndromo é uma palavra que é a mesma ao ser lida da esquerda para a direita ou da direita para a esquerda. Exemplos: raiar, arara. Usando uma pilha, escreva um algoritmo que leia uma palavra e verifique se é ou não um palíndromo.

12 ☠ Na lista A de exercícios, foi solicitada a elaboração de um algoritmo recursivo que imprime o n-ésimo termo da sequência de *Fibonacci*. Tente escrever o algoritmo iterativo equivalente, isto é, que simule as duas recursões com uma pilha explícita.

13. ☠ Aproveitando as ideias do exercício anterior, escreva um algoritmo iterativo com pilha que resolva o problema das Torres de Hanói de n discos com exatamente 2^n-1 movimentos.

C. Árvores

Importante:

- As soluções deverão gastar tempo proporcional ao número de nós da árvore.
- Nas questões 1 a 4, considere a seguinte estrutura de árvore binária:

```
struct node {
              int valor;
              node *esq, *dir;
            }
```

1. Para cada item abaixo, escreva uma função que recebe como parâmetro o ponteiro para a raiz de uma árvore binária.

a) Imprimir o seu percurso pré-ordem.

b) Imprimir o seu percurso in-ordem.

c) Imprimir o seu percurso pós-ordem.

d) Imprimir apenas o valor do primeiro e do último nó do seu percurso in-ordem.

e) Calcular a quantidade de nós dessa árvore.

f) Transformar essa árvore no seu *espelho*, sem fazer novas alocações.

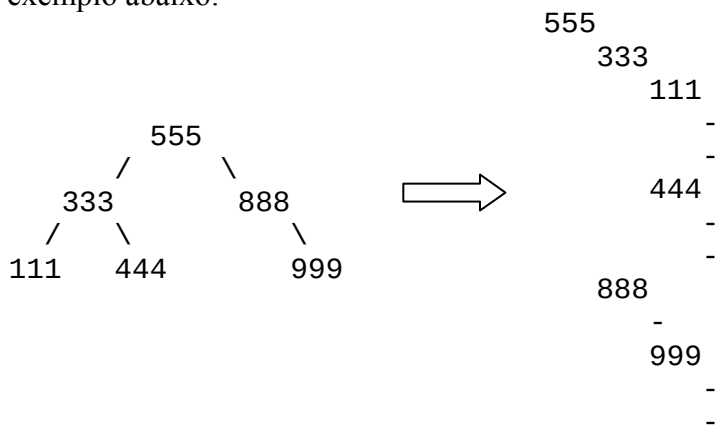
g) Verificar se um determinado valor **`k`** (também recebido como parâmetro) está ou não presente na árvore. *Observação*: Considere que a árvore não é de busca.

h) Verificar se essa árvore é ou não de busca.

2. Dados os ponteiros para a raiz **`T`** de uma árvore binária e para um determinado nó **`x`**, escreva uma função que retorne a profundidade de **`x`** nessa árvore.

Observação: Retorne -1 se **`T == null`** ou se **`x`** não estiver nessa árvore.

3. Escreva uma função que imprima os valores armazenados em uma árvore binária com recuos de margem proporcionais à profundidade do nó na árvore. Imprima '-' para representar **`null`**. Veja o exemplo abaixo:



4. Dada uma árvore binária de busca, escreva uma função que receba o ponteiro para a raiz dessa árvore e um valor **`x`**, presente na árvore, e devolva o sucessor de **`x`** na ordenação dessa mesma árvore.

5. Considere que em cada nó da árvore haja também o campo **`node *prox`**. Usando apenas uma fila que armazena **`node*`**, escreva uma função que recebe como parâmetro o ponteiro para a raiz de uma árvore binária e preenche o campo **`prox`** de todos seus nós, de tal maneira que corresponda a um percurso por nível nessa árvore.

D. Ordenação

1. Considere o vetor abaixo:

32	23	-3	0	12	20	-5

a) Simule nele a aplicação do algoritmo *HeapSort*, indicando passo a passo o conteúdo da estrutura de árvore do *heap*.

b) Idem para o algoritmo *MergeSort*, indicando os passos recursivos.

c) Idem para o algoritmo *QuickSort*, indicando os particionamentos.

2. a) Com o uso de uma pilha, escreva uma versão não-recursiva para o algoritmo *MergeSort*.

b) Nessa versão iterativa do *MergeSort*, poderia ser utilizada uma fila ao invés de pilha? Justifique.

c) Escreva uma versão não-recursiva do *MergeSort* que não utiliza nenhuma estrutura de dados auxiliar.

3. Escreva outras versões para a fase de particionamento do algoritmo *QuickSort*:

a) Onde o pivô é o último elemento do vetor.

b) Onde o pivô é escolhido através da mediana de três.

4. Sejam k vetores ordenados crescentemente, onde o número total de elementos desses k vetores é n. Descreva um algoritmo que realize o *merge* (isto é, a intercalação) de todos os vetores, produzindo um único vetor ordenado em tempo O(n.log k). Justifique.

5. Simule no computador uma instância do *QuickSort* com pior tempo de execução, até que o programa seja abortado por *overflow* na pilha de execução. Provoque essa situação nas duas versões do algoritmo (tenha em conta que serão instâncias diferentes):

a) Com duas chamadas recursivas (a pilha exigirá espaço O(n)).

b) Com uma chamada recursiva (a pilha exigirá espaço O(log n)).

6. Considere o seguinte jogo: uma pessoa pensa em um número inteiro na faixa de 1 a n, e outro tenta adivinhar esse número fazendo apenas perguntas do estilo "Esse número é maior (ou menor) do que x?".

a) Elabore uma boa estratégia para descobrir o número pensado, fazendo a menor quantidade possível de perguntas. Quanto tempo gastará a sua estratégia?

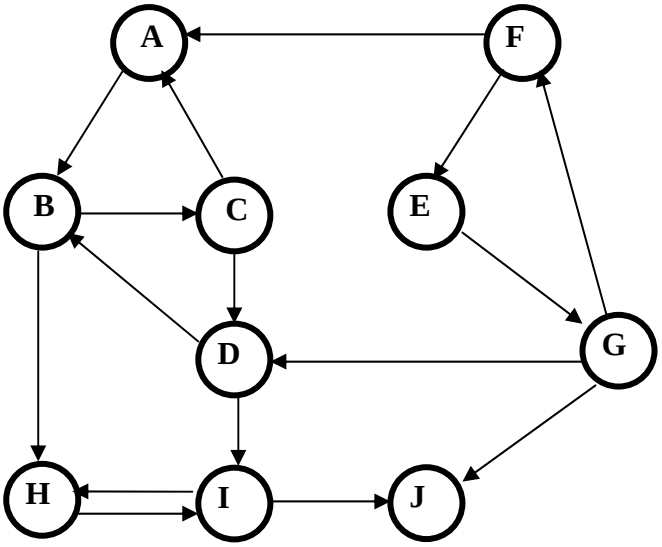
b) Considere agora o caso em que o intervalo é desconhecido, ou seja, você somente sabe que o número pensado é inteiro e positivo. Qual seria a sua nova estratégia, e quanto tempo gastará?

7. Seja S uma sequência aleatória de n números inteiros distintos. Uma inversão em S é um par de elementos x e y tais que x aparece antes de y em S, mas x > y. Por exemplo, na sequência (3, 4, 1, 6, 2, 5) há 6 inversões. Elabore um algoritmo de complexidade O(n.log n) que leia uma sequência S e determine a sua quantidade de inversões.

E. Grafos

Nas questões 1 a 7 a seguir, suponha que os grafos estejam armazenados através de **lista de adjacências**, onde tanto o vetor de vértices como as listas de nós adjacentes **obedecem à ordem alfabética**. Ao simular os algoritmos, **deixe indicado as eventuais informações associadas aos vértices e às arestas.** Quando for modificar alguma dessas informações, **não a apague**: apenas risque e escreva ao lado o novo conteúdo.

Nas questões 1, 2 e 3 abaixo, considere o seguinte digrafo:

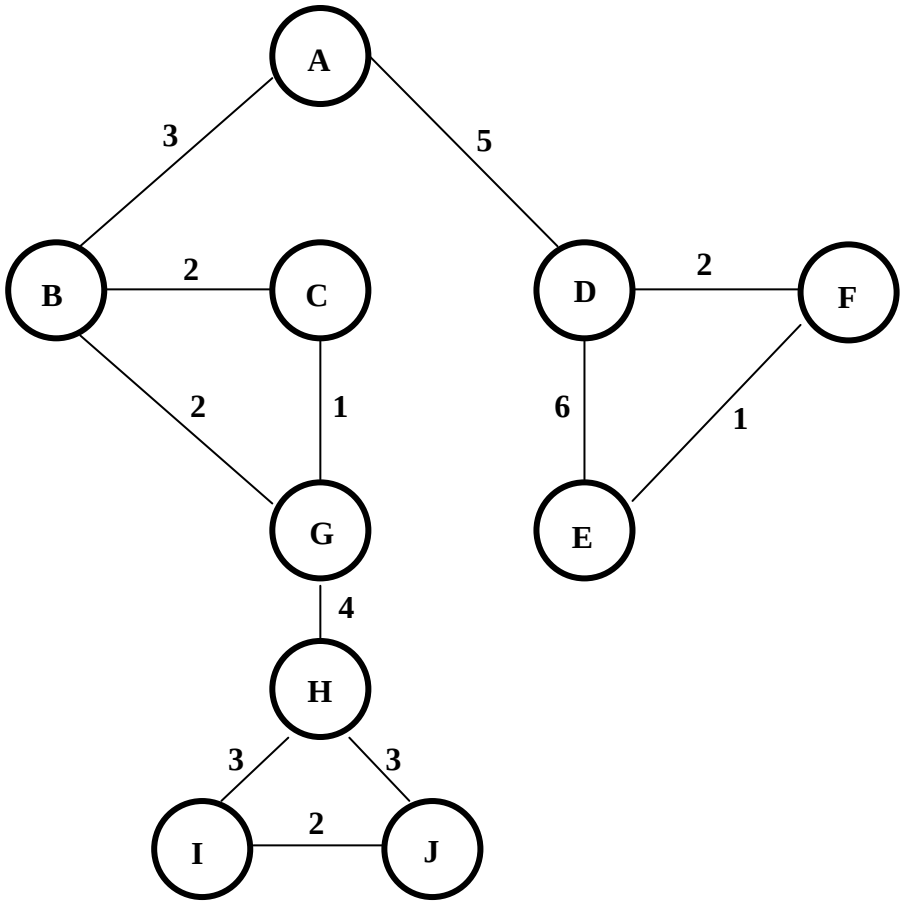


1. Através do algoritmo de Tarjan, classifique os arcos desse digrafo (T, F, B ou C) e desenhe a sua correspondente árvore de exploração. Qual é o primeiro ciclo encontrado?

2. Aplique nesse digrafo uma variante do algoritmo de Tarjan para encontrar os seus componentes fortemente conexos. Desenhe o correspondente digrafo reduzido (que é um DAG).

3. Desconsiderando as orientações das arestas desse digrafo, aplique uma variante do algoritmo de Tarjan para encontrar seus vértices e arestas de corte.

Considere o grafo abaixo nas questões 4, 5 e 6:

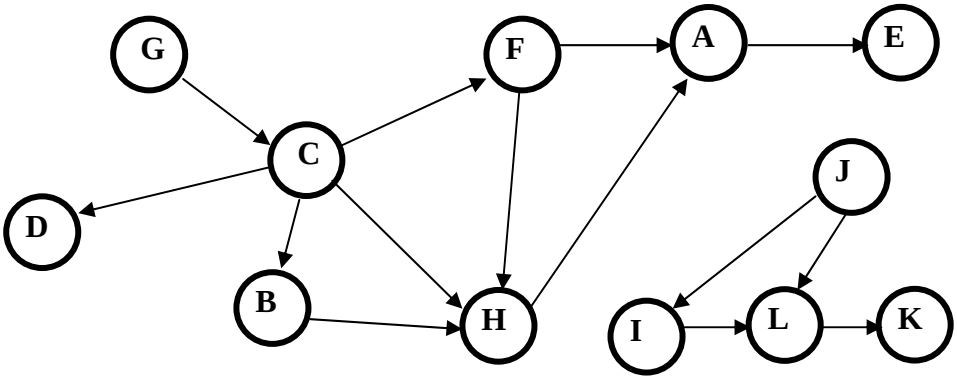


4. Aplique uma variante do algoritmo de Tarjan para encontrar os seus vértices e arestas de corte.

5. Através do algoritmo de Prim, encontre a árvore de espalhamento de custo mínimo.

6. Através do algoritmo de Dijkstra, encontre as distâncias mínimas do vértice A.

7. Encontre uma ordenação topológica para o digrafo abaixo:
a) através de um algoritmo baseado nos graus de entrada dos vértices;
b) através de uma exploração em profundidade.



8. Considerando um grafo não orientado, armazenado através de listas de adjacências, responda as perguntas abaixo:

a) Como poderia ser implementado um *algoritmo simples* que identifica seus componentes conexos?

b) Se o grafo for desconexo, como poderiam ser implementados os algoritmos que encontram seus vértices e arestas de corte?

9. Considere armazenamento de grafos através de lista de adjacências. Elabore algoritmos que realizam as tarefas abaixo:

a) Eliminação de eventuais arestas múltiplas e laços.

b) Dado um digrafo, criação do DAG correspondente aos seus componentes fortemente conexos.

c) Dado um grafo, criação do seu grafo complementar.

d) Dado um subconjunto S dos seus vértices, criação do subgrafo induzido por S.

10. Considere um grafo G de n vértices $\{v_1, ..., v_n\}$, armazenado através de uma matriz de adjacências A.

a) Seja I a matriz identidade de ordem n. O que representa (A *or* I) *and* (A *or* I), se os operadores lógicos *or* e *and* forem aplicados de modo análogo à soma e ao produto de matrizes, respectivamente?

b) Seja $C = (c_{ij})$ a matriz conectividade de G, isto é, $c_{ij} = 1$ se houver um caminho de v_i a v_j em G, ou $c_{ij} = 0$ em caso contrário, para $1 \leq i, j \leq n$. Com a ideia do item anterior, elabore um algoritmo de tempo $O(n^3.\log n)$ para o cálculo de C.

11. Dada a matriz conectividade de um grafo G de n vértices, como podem ser obtidos os seus componentes conexos? Qual a complexidade de tempo desse algoritmo? E se G fosse um digrafo, ou seja, como encontrar os seus componentes fortemente conexos?

12. Um digrafo ponderado G de n vértices $\{v_1, ..., v_n\}$ pode ser representado através de uma matriz de custos $M = (m_{ij})$, onde $m_{ij} = \text{custo}(v_i, v_j)$ se houver uma aresta de v_i a v_j em G, ou $m_{ij} = \infty$ em caso contrário, para $1 \leq i, j \leq n$.

Elabore um algoritmo que calcule a matriz $D = (d_{ij})$ de custo mínimo, onde d_{ij} será a distância entre v_i e v_j em G, para $1 \leq i, j \leq n$. Caso não exista um caminho entre os vértices v_i e v_j em G, então $d_{ij} = \infty$. Qual a complexidade de tempo desse algoritmo?

Sugestão: Pense numa solução análoga à da questão 10, utilizando os operadores *mínimo* e *soma*.

13. Dado um grafo ponderado, a distância entre dois vértices é o menor custo de um caminho que os une; quando não há caminhos entre eles, essa distância é infinita. Deseja-se calcular o diâmetro de um grafo ponderado, isto é, a maior distância entre dois vértices quaisquer. Mostre uma maneira eficiente de resolver esse problema.