CTC-12

LISTA DE EXERCÍCIOS

1) Preencha a tabela abaixo com $\in$ ou $\notin$:

	$\omega(\log n)$	$\Theta(n)$	$O(n \log n)$	$\Omega(n^2)$	$o(n^3)$
$6n^3 + 12n^2 + 12.\log n + 3$					
$4n + 3n.\log n + \log n$					

2) Dê um exemplo de função que seja simultaneamente:

a) $O(n^2)$ e $\Omega(\log^2 n)$

b) $\omega(\log n)$ e $o(n)$

c) $\Theta(n)$ e $\Omega(n^3)$

d) $o(2^n)$ e $\Omega(n^{50})$

e) $O(n^2)$ e $\omega(n.\log n)$

Se em algum caso não existir tal função, explique por quê.

3) Prove por indução matemática: $5^{n+1} + 2.3^n + 1$ é divisível por 8, para $n \geq 0$.

4) Prove por indução matemática: $2^n < n!$, para $n > 3$.

5) Calcule as complexidades de tempo e de espaço do programa abaixo, cujo corpo principal consiste apenas na chamada de **`Rec(n)`**:

```
Rec(n) {
    if (n==1) return 1;
    return 3*Rec(n-1) + Funcao(n);
}

Funcao(n) {
    for (i=0; i<n; i++)
         for (j=i; j>0; j--)
             v[i,j] = 4*i*i+j;
    return v[n-1,n-1];
}
```

Importante:

a) Considere que as operações básicas (multiplicações, subtrações, comparações, etc.) gastam tempo constante.

b) O vetor **`v`** tem dimensões `[0..n-1, 0..n-1]` e é uma variável global.

6) O algoritmo abaixo realiza a busca binária de **chave** em um vetor **v** de índices **0** a **N-1**, ordenado crescentemente:

```
int PesquisaBinaria (int v[], int chave, int N)
{
     int inf, sup, meio;
     inf = 0;
     sup = N-1;
     while (inf <= sup) {
          meio = (inf+sup)/2;
          if (chave == v[meio]) return meio;
          if (chave < v[meio]) sup = meio - 1;
              else inf = meio + 1;
     }
     return -1;   // não encontrado
}
```

Com o uso de recorrências, demonstre que a sua complexidade de pior caso é $\Theta(\log N)$.

Observação: basta considerar o caso em que **N** é potência de 2.

7) Dado o algoritmo recursivo abaixo, calcule o seu tempo exato de execução.

```
Rec(n) {
    if (n==1) return 1;
    return Rec(n-1) + Rec(n-1) + Rec(n-1) + 1;
}
```

Importante: considere que as operações básicas (multiplicações, subtrações, comparações, etc.) gastam tempo constante.

8) William G. Horner sugeriu uma maneira alternativa de exprimir polinômios. Por exemplo, o polinômio $p(x) = 3x^3+8x^2-4x+9$ teria o seguinte formato de Horner: $p(x) = 9+x(-4+x(8+x.3))$.

Dado um polinômio p(x) de grau n e um parâmetro x, calcule o número de multiplicações e adições necessárias para avaliá-lo:

a) através da sua expressão convencional;

b) através do formato de Horner.

9) Escreva algoritmos recursivos que recebam como parâmetro um número inteiro x e calculem o polinômio $p(x) = \sum a_k x^k$, $0 \leq k \leq n$:

a) através da sua expressão convencional;

b) através do formato de Horner.

Suponha que os coeficientes a_k estejam armazenados em um vetor de dimensão n+1.

10) Descreva detalhadamente o que é um *heap*.

11) Explique as ideias do algoritmo *QuickSort* e indique a ordem do pior caso.

12) Explique as ideias do algoritmo *MergeSort* e indique a ordem do pior caso.

13) Descreva os passos da ordenação crescente do algoritmo *RadixSort* para os seguintes números: 454, 901, 15, 23, 188, 901, 0, 675, 227, 6.

14) Foi visto que uma **fila de prioridades** (de máximo) é uma estrutura com as seguintes funcionalidades: *Max(), ExtractMax(), Insert(x)* e *Modify(k,x)*. Também vimos que pode ser eficientemente implementada com um *heap*. Por outro lado, se quiséssemos implementá-la com uma lista ligada, explique como elaboraria essas 4 funcionalidades e qual seriam as suas complexidades de tempo.

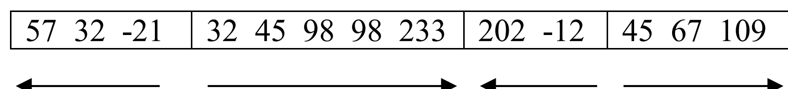
15) Sem utilizar o comando `for(;;)` (e nenhum outro comando de repetição), escreva um algoritmo recursivo que calcule o produto de duas matrizes quadradas A e B de ordem n, armazenando o resultado numa matriz C. Suponha que essas matrizes sejam variáveis globais, e que C esteja inicialmente zerada.

16) O produto interno de dois vetores n-dimensionais $u = (u_1, ..., u_n)$ e $v = (v_1, ..., v_n)$ é definido como $\sum u_i.v_i$. Escreva um algoritmo recursivo que, dados dois vetores n-dimensionais, calcule o seu produto interno.

17) Explique para que serve uma árvore AVL. Dê um exemplo de aplicação prática.

18) Suponha que um dado vetor de n elementos seja "tetratônico", isto é, seja formado por 4 subsequências ordenadas, cujos comprimentos são desconhecidos.

Veja um exemplo a seguir, onde n = 13 (as setas indicam a ordenação):



57 32 -21	32 45 98 98 233	202 -12	45 67 109
←	→	←	→

Para vetores desse estilo, descreva um algoritmo de ordenação que seja ótimo. Justifique.

19) Dado um vetor não ordenado de tamanho n, apresente um algoritmo que encontre o maior e o menor elementos, mas realizando no máximo $3n/2$ comparações.

20) Suponha um vetor de n posições cuja primeira metade está em ordem crescente. Para completar a ordenação desse vetor, os elementos da segunda metade são inseridos na parte ordenada, um a um, utilizando-se busca binária. Essa ordenação pode ser terminada em tempo $O(n.\log n)$? Justifique.

21) Prove por indução que uma árvore binária completa com $n = 2^k$ folhas tem altura $k = \log_2 n$.

22) Monte o autômato de busca do padrão "**bacabacb**".

23) Dado o texto "**Faturei com folga na prova de complexidade de algoritmos**" e o padrão "**algo**", indique (e justifique) o *número de comparações* que os algoritmos abaixo realizam até encontrarem esse padrão no texto:

a) Knuth-Morris-Pratt;

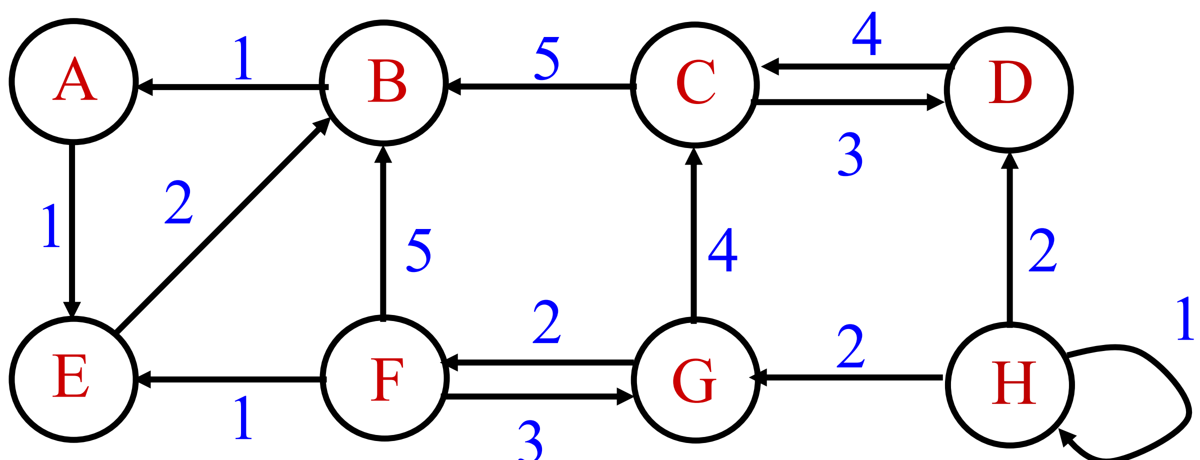
b) Boyer-Moore.

24) Dado o texto "**cababaabababa**" e o padrão "**ababa**", indique (e justifique) o *número de comparações* que os dois algoritmos abaixo realizam até encontrarem esse padrão no texto:

a) Knuth-Morris-Pratt;

b) Boyer-Moore.

O digrafo abaixo será utilizado nas três próximas questões.



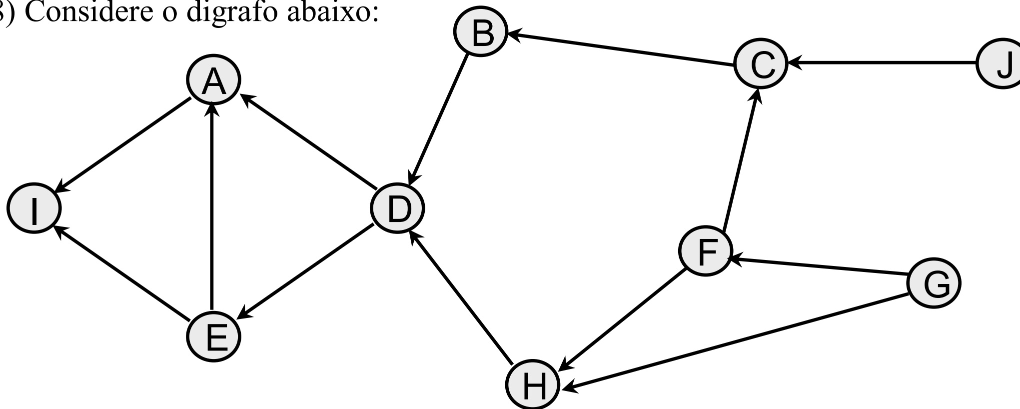
25) Através do algoritmo de Dijkstra, encontre o caminho mínimo entre os vértices H e A. Deixe indicado os passos que seguiu.

26) Através do algoritmo de Tarjan, encontre as componentes fortemente conexas. Deixe indicados os passos que seguiu.

27) Através do algoritmo de Prim, encontre uma árvore geradora de custo mínimo a partir do vértice A. Deixe indicado os passos que seguiu.

Importante: ignore a orientação dos arcos.

28) Considere o digrafo abaixo:



a) Supondo armazenamento por listas de adjacências, onde os vértices estão em ordem lexicográfica, aplique o algoritmo de Tarjan a partir do vértice G, classificando os arcos em 4 conjuntos diferentes: *Árvore*, *Avanço*, *Retorno* e *Cruzamento*. Algum desses conjuntos ficou vazio? Por quê?

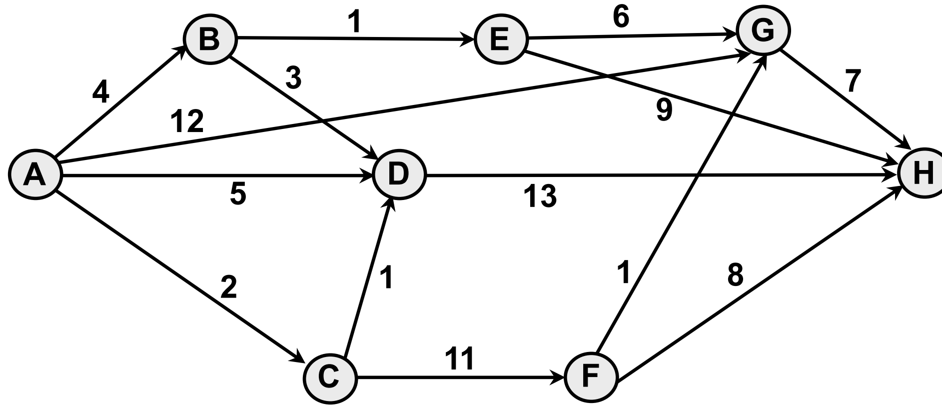
b) Encontre uma ordenação topológica para ele, caso exista. Justifique.

29) Ainda para esse mesmo digrafo, desconsidere as orientações das arestas e aplique a variante do algoritmo de Tarjan a partir do vértice A para encontrar seus vértices e arestas de corte.

30) Em aula foi apresentada uma implementação do algoritmo de Prim que utiliza uma estrutura de *heap*. Se ao invés dessa estrutura utilizássemos uma lista ligada, na qual os vértices fossem mantidos em ordem crescente de custo, qual seria a complexidade de tempo do algoritmo? Justifique.

31) Para grafos esparsos (com pouquíssimas arestas), qual é a melhor estrutura de armazenamento? Justifique. E para grafos densos?

32) Através de algum algoritmo visto em aula, encontre uma ordenação topológica para o digrafo abaixo. Indique os passos que seguiu.

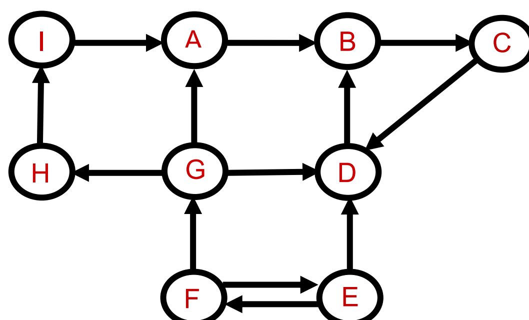


33) Para o digrafo da questão anterior, encontre as distâncias mínimas a partir do vértice A utilizando o algoritmo de Dijkstra. Indique os passos que seguiu.

34) Ainda para esse mesmo digrafo, encontre o fluxo máximo entre os vértices A e H simulando os passos do algoritmo de Edmonds-Karp.

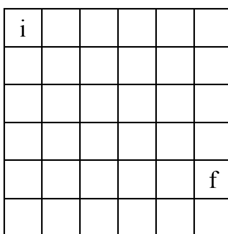
35) Desconsidere a orientação dos arcos desse mesmo digrafo e encontre uma árvore geradora de custo mínimo. Para isso, simule o algoritmo de Kruskal, indicando seus passos.

36) Com o algoritmo de Tarjan aplicado a partir do vértice I, classifique os quatro tipos de arcos do digrafo abaixo. Considere armazenamento em ordem lexicográfica.

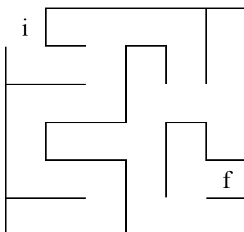


37) Com o algoritmo de Tarjan aplicado a partir do vértice I, encontre as componentes fortemente conexas do digrafo anterior. Considere armazenamento em ordem lexicográfica.

38) Uma grade pode ser representada através de um grafo: cada vértice seria uma posição dessa grade, onde uma aresta une posições vizinhas. Deste modo, os quatro vértices de canto têm grau 2, os demais vértices de borda têm grau 3, e o restante possui grau 4. Veja um exemplo abaixo com 36 vértices, onde estão indicados os vértices i e f, ambos de borda:



Dada uma grade e dois vértices de borda i e f, utilizando as operações *Union-Find* vistas em aula, mostre como essa grade pode ser transformada em um **labirinto aleatório sem ciclos**, onde i é a sua entrada e f a sua saída. Veja abaixo um possível labirinto para o exemplo anterior:



39) Escreva um novo algoritmo para encontrar as componentes conexas de um grafo, baseado nas operações *Union-Find* vistas em aula. Supondo que essas operações sejam implementadas com árvores e compressão de caminhos, calcule a complexidade de tempo deste algoritmo.

40) Escreva um algoritmo *Divisão-e-Conquista* para encontrar o valor máximo do vetor A[1..n] de inteiros. Calcule a sua complexidade.

41) Através de *Programação Dinâmica*, escreva um algoritmo que, dado n, monte uma matriz de ordem n com os valores do triângulo de Pascal. Calcule a sua complexidade.

Exemplo para n = 6:

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

42) Dado o texto abaixo, encontre uma codificação de Huffman:

AABCCDAABAEAEDBCCEEAAA

43) Dada o produto de matrizes abaixo, simule o algoritmo de *Programação Dinâmica* que encontra a melhor parentização:

$P = M_1 . M_2 . M_3 . M_4 . M_5$

Respectivas dimensões das matrizes M_i: (2x3), (3x7), (7x8), (8x4), (4x1).

44) Calcule o produto 34 x 1729 através do algoritmo de Karatsuba.

Importante: basta simular apenas o primeiro nível de recursão.

45) Através do algoritmo de *Programação Dinâmica* visto em aula, encontre a maior subsequência comum de <AACGTGGCCTA> e <CACGTTCCA>.

46) Sejam X[1..n] e Y[1..m] vetores de caracteres, onde n e m são inteiros positivos. Considere X vazio quando n=0, e Y vazio quando m=0.

a) Baseado no algoritmo de *Programação Dinâmica* visto em aula, escreva um pseudocódigo *memoization* que encontre o **tamanho da maior subsequência comum** de X e Y.

b) Neste problema, *memoization* pode ter melhor desempenho que *Programação Dinâmica*? Explique.

47) O algoritmo abaixo gera aleatoriamente uma permutação entre os números 0 e n-1:

```
Fisher-Yates (n) {
      for i = 0 to n-1 do
            p[i] = i;
      for i = n-1 to 1 do {
            j = random(0,i);  // retorna um número aleatório entre 0 e i
            p[i] ↔ p[j];
      }
      return p;
}
```

Supondo que a função **random** gasta tempo constante, qual é a complexidade de tempo deste algoritmo? Demonstre que ele gera todas as permutações com a mesma probabilidade.

48) Dizemos que um algoritmo *Las Vegas* tem tempo $\tilde{O}(f(n))$ se seu tempo de execução é limitado por $c\alpha f(n)$ para todo $n > n_0$ com probabilidade mínima $1-n^{-\alpha}$, onde c e n_0 são constantes.

Dado um vetor A[0..n-1], onde apenas um dos seus elementos repete-se n/2 vezes e os outros n/2 elementos são distintos, o algoritmo abaixo identifica o elemento repetido. Demonstre que seu tempo é $\tilde{O}(\log n)$.

```
RepetidoLV(A,n) {
      repeat
            i = random(0,n-1);  // retorna um número aleatório entre 0 e n-1
            j = random(0,n-1);
            if (i != j) and (A[i] == A[j])
                  return A[i];
      until true;
}
```

49) Dado um grafo não orientado G(V,E), o *Problema da Cobertura de Vértices* consiste em encontrar o **menor** subconjunto $C \subseteq V$ tal que, se $(u,v) \in E$, então $u \in C$ ou $v \in C$.

Prove que o algoritmo abaixo encontra uma solução com fator de aproximação igual a 2.

```
CoverVertex1(V,E) {
      C ← ∅;
      while (|E| > 0) {
            Escolha uma aresta qualquer <u,v> ∈ E;
            Retire os vértices u e v de G;
            C ← C ∪ {u,v};
      }
      return C;
}
```

50) Considere o algoritmo guloso abaixo para o *Problema da Cobertura de Vértices*. Mostre que seu fator de aproximação é O(log |V|).

```
CoverVertex2(V,E) {
      C ← ∅;  G0 ← G(V,E);  i ← 0;
      while (Gi tiver pelo menos uma aresta) {
            i ← i+1;
            vi ← vértice de Gi-1 com grau máximo di;
            Gi ← Gi \ vi;      // retira vi e suas arestas incidentes
            C ← C ∪ {vi};
      }
      return C;
}
```