Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Programa de Pós-Graduação em Ciência da Computação
Projeto e Análise de Algoritmos - 1ºsemestre de 2010
Professor: David Menotti

Trabalho Prático 1

Valor: 1,0 pontos (10% da nota total) Data de Entrega: 02/05/2010

1. O algoritmo de Ordenação por Inserção pode ser expresso como um procedimento recursivo da seguinte forma: para ordenar $A[1..n]$, ordena-se recursivamente $A[1..n-1]$ e então insere-se $A[n]$ no vetor ordenado $A[1..n-1]$.

   (a) Implemente uma função recursiva em C/C++ para implementar o algoritmo descrito acima;

   (b) Escreva uma equação de recorrência para o tempo de execução dessa versão recursiva;

   (c) Resolva essa equação de recorrência, ou seja apresente a solução em forma fechada (função de complexidade)

   (d) Determine a ordem de complexidade desse algoritmo através do Teorema Mestre.

2. Descreva um algoritmo e implemente uma função em C/C++ com complexidade de tempo $\Theta(n \log n)$ que, dado um conjunto S de n inteiros e um outro inteiro x, determina se existe ou não dois elementos de S cuja soma é exatamente x.

3. (*Limite Inferior*) Para cada um dos problemas elencados abaixo: (1) Apresente um algoritmo ótimo para resolver esse problema; (2) Prove que o algoritmo apresentado é ótimo.

   (a) Considere um arranjo A com n elementos não ordenados. O problema é encontrar o maior valor dentre estes n elementos.

   (b) Considere um arranjo A com n elementos não ordenados. O problema é encontrar o maior e o menor valores dentres estes n elementos.

   (c) Considere um arranjo A com n elementos não ordenados. O problema é encontrar o maior e o segundo maior valor dentre estes n elementos.

   (d) São dados $2n$ elementos distintos distribuídos em dois arranjos A e B, cada um com n elementos ordenados, tal que $A[0] < A[1] < ... < A[n-2] < A[n-1]$ e $B[0] < B[1] < ... < B[n-2] < B[n-1]$. O problema é encontrar o n-ésimo maior valor dentre estes $2n$ elementos.

4. (*Ordenação por Inserção em pequenos vetores no Mergesort*) Sabe-se que o algoritmo Mergesort executa no pior caso em tempo $\Theta(n \log n)$ e o algoritmo de ordenação por Inserção no pior caso em tempo $\Theta(n^2)$. No entanto, os fatores constantes do Inserção o tornam mais rápido que o Mergesort para um pequeno valor de n. Assim, faz sentido usar o algoritmo de Ordenação por Inserção quando os sub-problemas tornam-se suficientemente pequenos. Considere a seguinte modificação no Mergesort: n/k sub-listas de comprimento k são ordenadas usando o algoritmo de ordenação por Inserção e então combinadas/intercaladas (*merged*) usando o mecanismo do Mergesort, sendo k o valor a ser determinado.

   (a) Mostre que se as n/k sub-listas, cada uma de comprimento k, podem ser ordenadas pelo algoritmo de Ordenação por Inserção no pior caso em tempo $\Theta(nk)$

   (b) Mostre que as sub-listas podem ser combinadas (merged) no pior caso em tempo $\Theta(n \log n/k)$

   (c) Dado que o algoritmo modificado executa no pior caso em tempo $\Theta(nk + n \log n/k)$, qual é o maior valor assintótico (usando notação Θ de k como uma função de n para o qual o algoritmo modificado tem o mesmo tempo de execução assintótico do Mergesort padrão?

   (d) Na prática, como o valor de k seria escolhido?

   (e) Compare o tempo de execução do algoritmo modificado com os algoritmos clássicos Mergesort e Inserção.

5. (*Inversões*) Seja $A[1..n]$ um vetor com n número distintos. Se $i < j$ e $A[i] > A[j]$, então o par (i, j) é chamado de uma inversão de A.

   (a) Liste as cinco inversões do vetor $< 2, 3, 8, 6, 1 >$;

   (b) Que vetor com elementos do conjunto $\{1, 2, ..., n\}$ tem o maior número de inversões? Quantas inversões existem?

   (c) Qual é a relação entre o tempo de execução do algoritmo de ordenação por Inserção e o número de inversões do vetor de entrada? Justifique sua resposta.

   (d) Apresente um algorimto que determina o número de inversões em qualquer permutação de n elementos no pior caso em tempo $\Theta(n \log n)$. (Dica: modifique o Mergesort).

6. **Fatorial de Números Grandes** - Faça um programa que permita calcular o fatorial de números relativamente grandes como o fatorial de 10000. Você não deve usar qualquer biblioteca de funções da linguagem C++. O problema deve ser resolvido usando apenas a memória principal com a menor quantidade possível de espaço. O objetivo deste trablaho é estudar a complexidade de espaço. Procure implementar também a operação de multiplicação da forma mais eficiente possível. Procure na literatura algoritmos eficientes para multiplicação de números inteiros.

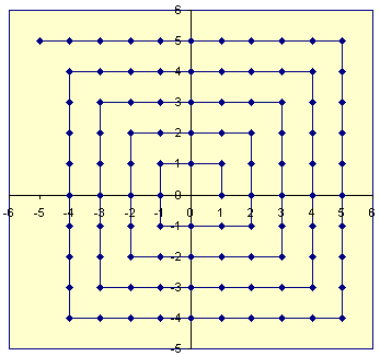
   Sugestão: use um *nibble* (metade de um octeto ou *byte*) para armazenar um algarismo decimal ou melhor ainda, algarismo hexadecimal.

7. **Decomposição de Números** - Faça um programa recursivo para gerar a decomposição de um número inteiro positivo na soma de todos os possíveis fatores como mostrado a seguir. Por exemplo, para $n = 5$, temos:

   5
   $4 + 1$
   $3 + 2$
   $3 + 1 + 1$
   $2 + 2 + 1$
   $2 + 1 + 1 + 1$
   $1 + 1 + 1 + 1 + 1$

8. **Espiral Quadrada** - Seja a espiral quadrada como apresentada abaixo. Faça um programa que apresente as coordenadas (x, y) de um dado ponto n fornecido na entrada. Apresente três algoritmos distintos que executam no pior caso em:

   (a) $O(1)$
   (b) $O(\sqrt{n})$
   (c) $O(n))$
   (d) Você conhece algum problema "não usual" que tenha algoritmos com complexidades tão diferentes como o da Espiral Quadrada? Se sim, enuncie esse problema e indique os algoritmos e/ou referências para sua solução.

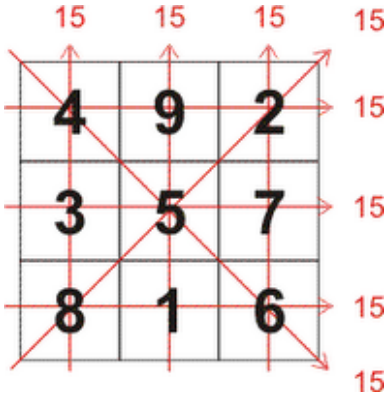


9. **Máxima Soma** - Dado um vetor n de números inteiros, determine a máxima soma encontrada em um sub-vetor contíguo desse vetor. Se todos números forem negativos assumir que a soma vale 0. A figura abaixo à esquerda mostra um vetor com 10 elementos. Nesse caso, a máxima soma é 20, dada pela soma dos elementos contíguos do sub-vetor de índices de 3 a 7, como mostrado na figura à direita.

   Tente apresentar um algoritmo com custo de execução menor que $O(n^2)$.

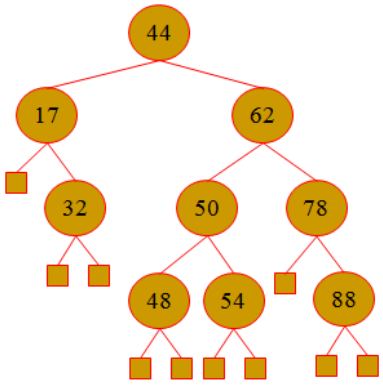
10. **Quadrado Mágico** - Quadrado Mágico é um quadrado de lado n, onde a soma dos números das linhas, das colunas e das diagonais é constante. Em cada posição do quadrado pode-se colocar um número entre 1 e n^2, sendo que cada número só pode aparecer uma única vez no quadrado.

    A figura abaixo mostra uma possível solução para o quadrado mágico de lado $n = 3$.



    Gere o quadrado mágico para $3 \leq n \leq 10$.

11. **Caminhamentos em Árvore Binária de Busca**



    (a) Escreva um procedimento recursivo baseado em um dos caminhamentos em árvore binária para calcular a altura de uma árvore binária de busca (ou pesquisa).

    (b) Em cada entrada da tabela abaixo, diga SIM se a combinação linha/coluna é sempre verdadeira e NÃO, caso contrário.

    A expressão $x \prec y$ significa que x precede y no caminhamento em questão. Um nó está à esquerda ou à direita de outro se e somente se eles têm um ancestral em comum.

    (c) O caminhamento por nível de uma árvore primeiro lista a raiz, depois todos os nós que estão no nível 1, depois todos os nós no nível 2, etc. Escreva um programa $O(n)$ (onde n é o número de nós na árvore) para listar todos os nós de uma árvore por nível (em cada nível, do nó mais à esquerda para o mais à direita).

	pré-ordem(n) $\prec$ pré-ordem(m)	in-ordem(n) $\prec$ in-ordem(m)	pós-ordem(n) $\prec$ pós-ordem(m)
n está a esquerda de m			
n está a direita de m			
n é um ancestral de m			
n é um descendente de m			

(d) Escreva um procedimento para listar todos os caminhos da raiz até os nós folhas.

O que deve ser entregue

- Código fonte dos programas em C ou C++ (bem identado e comentado).
- Documentação do trabalho.

Entre outras coisas, a documentação deve conter:

1. Introdução: descrição de cada problema a ser resolvido.
2. Implementação: descrição sobre a implementação do programa. Muito importante: os códigos utilizados nas implementações devem ser inseridos na documentação.
3. Análise de Complexidade: estudo da complexidade de tempo e espaço das funções implementadas.
4. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
5. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sítio da Internet se for o caso. Uma referência bibliográfica deve ser citada no texto onde é utilizada.
6. Em LaTeX: A documentação deve ser elaborada obrigatoriamente em LaTeX. Veja modelo de como fazer o trabalho em latex: `http://www.decom.ufop.br/menotti/paa101/tps/modelo.zip`
7. Formato final: mandatoriamente em PDF (`http://www.pdf995.com/`).

Como deve ser feita a entrega

A entrega DEVE ser feita via Moodle (`www.decom.ufop.br/moodle`) na forma de um único arquivo zipado, contendo o código fonte, arquivos diversos e a documentação. Também deve ser entregue a documentação impressa na próxima aula teórica após a data de entrega do trabalho.

Comentários Gerais

- A maioria destes problemas foram extraídos de [1, 4, 5, 2, 3] e as vezes de alguma forma alterados;
- Comece a fazer este trabalho/resolver esta liga logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- O trabalho é individual (grupo de UM aluno);
- Trabalhos copiados (e FONTE) terão nota zero. Devido a recorrentes problemas com cópias de trabalhos (plágios), os autores de trabalhos copiados também terão a maior nota dentre os testes teóricos levada a zero, como forma de punição e coação ao plágio acadêmico;
- Trabalhos entregues em atraso serão aceitos, todavia a nota atribuída ao trabalho será zero;
- Evite discussões inócuas com o professor em tentar postergar a data de entrega do referido trabalho.

Referências

[1] Antônio Alfredo Ferreira Loureiro. Sítio da disciplina de Projeto e Análise de Algoritmos do PPGCC/UFMG - 1ºsemestre de 2010, 2010. `http://homepages.dcc.ufmg.br/~loureiro/alg/101/`, visitado em 29/03/2010.

[2] A.M. Tenenbaum, Y. Langsam, and M.J. Augenstein. *Data Strcutures Using C*. Prentice-Hall International Editions, 1995.

[3] A.M. Tenenbaum, Y. Langsam, and M.J. Augenstein. *Estruturas de Dados Usando C*. Makron Books/Pearson Education, 1995.

[4] N. Ziviani. *Projeto de Algoritmos: com implementações em Pascal e C*. Cengage Learning (Thomson / Pioneira), São Paulo, 2nd edition, 2004.

[5] N. Ziviani. *Projeto de Algoritmos com implementações em Java e C++*. Cengage Learning (Thomson / Pioneira), São Paulo, 1st edition, 2006.