Algoritmos e Estruturas de Dados

Exercício Programa 1
Análise empírica de algoritmos recursivos

Rodrigo de Souza

Prazo: 20 de Maio de 2018 (23h55)

1 Enunciado

Sua tarefa neste EP é implementar alguns algoritmos recursivos e compará-los experimentalmente, ou seja, através da *contagem da quantidade de operações executadas*,[1] com versões iterativas para o mesmo problema. São eles (os códigos entre parêntesis indentificam cada um dos algoritmos que você vai implementar):

- Um algoritmo recursivo do tipo divisão-e-conquista (MAX-REC) e um algoritmo iterativo (MAX-IT) para encontrar o máximo de um vetor de inteiros.
- Um algoritmo recursivo do tipo divisão-e-conquista (CRESC-REC) e um algoritmo iterativo (CRESC-IT) para testar se um vetor de inteiros está em ordem crescente.
- Um algoritmo recursivo do tipo divisão-e-conquista (LOC-REC) e um algoritmo iterativo (LOC-IT) para encontrar a posição de um inteiro x em um vetor crescente de inteiros (como fizemos na aula, queremos encontrar o índice j tal que `v[j - 1] < x <= v[j]`).
- Um algoritmo recursivo do tipo divisão-e-conquista (SEG-REC) e um algoritmo iterativo (SEG-IT) para calcular calcular o segmento de soma máxima de um vetor de inteiros.

Alguns desses algoritmos vocês encontram no material de nosso guru P. Feofiloff (`https://www.ime.usp.br/~pf/algoritmos/`). Você pode copiar funções prontas, desse ou de outros lugares na Internet (cuidado para não copiar porcarias), mas neste caso deve mencionar a fonte explicitamente em um comentário.

O último problema, *segmento de soma máxima*, está descrito claramente no Capítulo 5 (página 27) do *Minicurso de Análise de Algoritmos* de P. Feofiloff, que vocês encontram em `http://www.ime.usp.br/~pf/livrinho-AA/AA-BOOKLET.pdf`. Vocês deverão implementar dois algoritmos que estão descritos em pseudocódigo nesse capítulo: SEG-IT é o algoritmo descrito na página 28, e SEG-REC está descrito na página 29. Vocês terão também uma explicação sobre esse problema e esses algoritmos na aula.

[1] Isso é um pequeno exemplo do que se chama *profiling* na área de Engenharia de Software. Se curioso veja `https://en.wikipedia.org/wiki/Profiling_(computer_programming)`.

1.1 Testes

Para cada item, você deve fazer vários testes com ambos os algoritmos, para diversos valores de n, o tamanho da instância.

Para realizar esses testes, seu programa deve gerar instâncias aleatoriamente[2] para cada valor de n; é muito melhor do que pedir para o usuário digitar todos os valores (então, *seu programa não pede nenhuma entrada para o usuário*). Para facilitar, defina duas constantes no seu programa: uma delas representa o passo para os diversos tamanhos n das instâncias, a outra representa o tamanho máximo de n. Por exemplo, se o tamanho máximo é 20000, e o passo é 5000, você vai gerar instâncias de tamanhos $5000, 10000, 15000, 20000$ (são poucas aqui; é bom gerar pelo menos vinte instâncias – também é bom usar tamanhos grandes, vetores de tamanho 10, 50, etc. podem não dizer muita coisa).

Para cada algoritmo, você deve fazer um gráfico indicando, nas abscissas, os tamanhos n das instâncias, e nas ordenadas, o número de operações realizadas. Assim, você deverá incluir no código de cada função um contador do número de operações (tipicamente, uma variável global). Antes de cada execução, o contador vale zero; para os algoritmos recursivos, o contador deve ser incrementado a cada chamada recursiva, e para os iterativos, você deve incrementá-lo na posição do código onde o algoritmo está efetivamente fazendo trabalho.

Para gerar o gráfico, você deve, para cada algoritmo, gerar um arquivo texto contendo duas colunas de números. Na primeira coluna seu programa grava o tamanho n da entrada, e na segunda coluna, o número de operações realizadas. De posse desse arquivo, você pode gerar um gráfico usando por exemplo o Excel ou a ferramenta gnuplot. Recomendo a segunda ferramenta, você pode baixá-la no endereço `http://www.gnuplot.info/`. Esse endereço tem também uma farta documentação da ferramenta gnuplot, mas se quiser uma receita rápida para traçar os gráficos, encontrei esse tutorial: `http://lowrank.net/gnuplot/datafile-e.html`. Identifique, nos gráficos, somente o código do algoritmo (aqueles descritos acima, no enunciado), não precisa escrever outras coisas (exceto os valores das abscissas e ordenadas).

Você deve elaborar um pequeno relatório, seguindo o formato de artigo da SBC, que você encontra em `http://www.sbc.org.br/documentos-da-sbc/category/169-templates-para-artigos-e-capitulos-de-livros`. Ou seja, seu relatório deve ter um título, a indicação do autor, um pequeno resumo, uma introdução, e referências bibliográficas (basta colocar os sites e livros que você consultou). No seu relatório, você vai, de forma organizada, visualmente agradável, estruturada, inserir os gráficos, com uma legenda para cada um, indicando qual é o algoritmo correspondente, e vai tirar suas conclusões após análise visual dos gráficos. Por exemplo, parece que os algoritmos recursivo e iterativo para um determinado problema são igualmente eficientes? Parece que um determinado algoritmo tem complexidade linear, ou é outra coisa? Tente estruturar seu relatório como um artigo científico.

Você deve entregar no AVA seu programa, um único fonte C, e o relatório em formato pdf. Você deve enviar os dois arquivos somente. Em particular, não entregue os arquivos texto usados para gerar os gráficos.

Seu trabalho ficará muito melhor se você usar LaTeX.

[2] Veja como em `https://www.ime.usp.br/~pf/algoritmos/aulas/random.html`.

2 Instruções gerais

- Seu programa deve ser feito em C. Se usar algoritmos oriundos de fontes externas, você deve mencionar explicitamente a fonte em um comentário.
- Seu programa deve consistir de um único arquivo, e deve ser organizado, modular, ou seja: *deve consistir de diversas funções que, juntas, realizam a tarefa descrita.* Um programa desorganizado (o código inteiro misturado dentro de uma única função `main` por exemplo) pode levar a descontos de nota.
- Cada função recebe somente os parâmetros descritos no problema, e devolve o valor especificado também no problema; não invente novos parâmetros e novos valores que fazem coisas estranhas ao problema.
- Documente cada função dizendo o quê ela faz. A documentação e a apresentação do seu programa poderão levar a descontos de nota se estiverem ruins. Você deve seguir as recomendações em `https://www.ime.usp.br/~pf/algoritmos/aulas/layout.html` e `https://www.ime.usp.br/~pf/algoritmos/aulas/docu.html`.
- Escreva no início do código um cabeçalho com comentários, indicando nome, número do EP, data, nome da disciplina.
- Seu relatório deve obrigatoriamente estar em pdf. Não entrege arquivos de outros formatos. Não entregue um arquivo compactado (você deve enviar somente dois arquivos, o código, e o relatório).
- O nome do arquivo fonte deve ser `nome_sobrenome_ep1.c` onde `nome` e `sobrenome` são respectivamente seu primeiro nome e seu último nome, e o do relatório segue mesmo modelo: `nome_sobrenome_ep1.pdf`.
- A entrega deverá ser feita no ambiente Moodle, no espaço reservado para essa finalidade. Consulte o ambiente para instruções. A entrega será eletrônica (não receberei exercícios impressos).

3 Correção

Não sou partidário da correção "booleana". Mesmo quando o exercício não está correto, busco ler o que o aluno fez e atribuir alguma nota. Meus critérios são os seguintes:

- Trabalhos copiados (relatório ou programa) são anulados.
- Um trabalho consistindo somento do código, sem o relatório, vale no máximo 5 (só vou julgar o código).
- Um programa que não compila vale no máximo 3 (neste caso, evidentemente é impossível haver um relatório, já que o programa nem funcionou). Neste caso, leio o código para tentar julgar as tentativas do aluno.
- Caso o programa compile, e o relatório tenha sido entregue, cada um vale 5 na nota.

- Um relatório mal escrito, desorganizado, recebe 2 pontos de desconto. Assim, capriche na apresentação. Use Português correto, frases curtas.

- Um programa mal escrito, desorganizado, recebe 2 pontos de desconto, *mesmo que funcione*. Ler o código leva muito tempo, mas tento fazer o melhor, lendo tudo o que vocês me entregam. Se o programa está organizado, documentado, a leitura é rápida. Se está confuso, às vezes é muito difícil compilar na cabeça o que o aluno faz, e isso leva muito tempo, então, *não vou compilar na cabeça códigos difíceis de ler*. É por isso que esses códigos recebem desconto. Daí a importância de caprichar no código também (veja dicas na seção anterior).

4 Plágio

O problema do plágio de código é endêmico em disciplinas que envolvem programação. Essa prática será fortemente combatida. Você está se enganando se copiar o programa de seu colega. O maior prejudicado é você mesmo.

Neste semestre, vou usar uma ferramenta automática para detecção de plágios em código. Essa ferramenta faz uma análise de um conjunto de arquivos fonte (os EP's) e emite um relatório, indicando a semelhança entre pares de arquivos. Em particular, essa ferramenta (que já tem anos de extrada, e uma *expertise* subjacente) detecta tentativas de burlar o plágio como mudança de nomes de variáveis e alterações na indentação. Ou seja, é muito difícil copiar e não ser descoberto. E infelizmente preciso ser bastante rigoroso caso a ferramenta detecte cópias (veja critérios de correção acima).

Vocês podem e devem pensar em soluções em conjunto. Mas uma vez encontrada a solução, cada um faz seu programa. O máximo que podem fazer é, eventualmente, trocar pequenos trechos sem importância para a solução, por exemplo, código para entrada e saída. E caso o façam, devem indicar com comentários ("esse trecho foi desenvolvido em conjunto com fulano de tal", etc.).

Ao copiar, você também prejudica seu colega que fez o código, porque ambos os trabalhos serão anulados. Melhor evitar essa prática. Se não fez o exercício, melhor não entregar e se concentrar no próximo.