Estruturas de Dados — MC202ABC
1º Semestre 2018

Instituto de Computação — UNICAMP

2ª Lista de Exercícios

Entregue os exercícios até 06/05/2018 pelo SuSy.
Leia com cuidado as instruções abaixo.

Instruções:

- Nesta lista, escolha **apenas três** dos cinco exercícios propostos para resolver. Exercícios resolvidos a mais não contarão como nota extra.
- Os exercícios deverão ser entregues separados em tarefas diferentes no SuSy: o exercício 1 deve ser entregue na tarefa `l2ex1`, o exercício 2 deve ser entregue na tarefa `l2ex2` e assim por diante. Caso não faça algum dos exercícios, basta não submeter na tarefa referente ao mesmo.
- Se preferir, apresente algoritmos como um pseudocódigo ao invés de uma implementação em C, isto é, descreva o seu algoritmo com uma lista de passos muito bem definidos, inclusive usando estruturas como **if** e **while**.
- A nota da lista será dada em relação aos três exercícios resolvidos, apesar da submissão ocorrer em três tarefas diferentes do SuSy.
- Os exercícios podem ser digitados em um computador (preferencialmente) ou serem escritos à mão e digitalizados (você pode inclusive tirar uma foto). Porém, a entrega precisa ser feita necessariamente por meio de um arquivo **PDF** com tamanho de no máximo **2MB**. Outros formatos *não serão aceitos*.
- Garanta que o documento tenha qualidade o suficiente para ser lido. Documentos escritos à mão devem ter letras legíveis e feitos preferencialmente à caneta, já que o lápis tem pouco contraste para a digitalização. Assim, se possível, opte por fazer o exercício no computador ao invés de digitalizar.
- Caso tenha problemas com o tamanho do arquivo gerado, procure por um compressor de PDFs online. Existem vários serviços que recebem um PDF e geram o mesmo PDF com um tamanho menor. Porém, tome cuidado de garantir que o arquivo continue legível.
- Lembre-se de indentar corretamente o seu código para facilitar o entendimento.
- **Não se esqueça de colocar nome e RA em cada exercício.**
- A correção de cada exercício será enviada para o seu email institucional (da DAC). Garanta que há espaço na sua cota para receber os emails[1] e verifique o spam se necessário.

[1] Considere fazer o redirecionamento do seu email da DAC para o seu email pessoal: https://www.dac.unicamp.br/portal/estudantes/webmail-mais-informacoes.

Exercício 1 (`l2ex1`): Dado um vetor V com n inteiros que pertencem ao intervalo $[0, k]$.

a) Dos algoritmos de ordenação vistos em sala, qual seria o mais apropriado para ordenar o vetor V? Justifique a escolha apresentando a complexidade de tempo em notação O.

b) Para quais valores de k é possível ordenar o vetor V em tempo linear, ou seja, $O(n)$?

c) Sabendo que $k = n^l - 1$. Desenvolva um algoritmo com tempo $O(n \times l)$ para ordenar o vetor V. Justifique o tempo do algoritmo desenvolvido. **Não** é necessário desenvolver o algoritmo em C, basta apresentar um pseudocódigo com a ideia.

Exercício 2 (`l2ex2`): Um heap d-ário é um heap onde cada elemento tem até d filhos. Nesta questão, queremos representar um heap d-ário máximo usando um vetor D.

a) Dado um índice i, qual o índice de D corresponde ao pai de i em um heap d-ário? E quais índices correspondem aos d filhos de i?

b) Chamamos de heap terciário o caso particular de heap d-ário com $d = 3$. Dado um vetor D com n elementos, dê um algoritmo $O(n)$ que cria um heap terciário máximo. **Não** é necessário desenvolver o algoritmo em C, basta apresentar um pseudocódigo com a ideia. Argumente, baseado no que foi visto em sala, que o algoritmo dado possui tempo $O(n)$.

Exercício 3 (`l2ex3`): Na busca binária, temos um vetor V ordenado e, a cada iteração, dividimos V em duas partes de tamanhos "iguais" e procuramos o elemento em apenas uma das partes. Dado um vetor ordenado V de tamanho n, desenvolva uma busca ternária em V, ou seja, um algoritmo de busca que divide o vetor em **três** partes de tamanhos "iguais".

Exercício 4 (`l2ex4`): Dado um conjunto de n listas $E = \{E_1, E_2, ..., E_n\}$, em que cada lista E_i possui n_i inteiros ordenados e seja $N = \sum_{i=1}^{n} n_i$. Dê um algoritmo com complexidade $O(N \ln |E|)$ que recebe o conjunto E como parâmetro e devolve um vetor que contém todos os elementos das listas de E ordenados. **Não** é necessário desenvolver o algoritmo em C, basta apresentar um pseudocódigo com a ideia. **Não** é necessário apresentar a implementação das estruturas de dados utilizadas, apenas o seu uso no algoritmo.

Exercício 5 (`l2ex5`): Um *loop* é chamado de DOALL[2] quando não possui dependência entre suas iterações. Esse tipo de *loop* é útil quando desejamos paralelizar um algoritmo, já que podemos executar várias iterações ao mesmo tempo. Um *loop* que necessita do estado gerado por iterações anteriores para executar a iteração atual é chamado de DOACROSS.

Exemplo de *loop* DOALL:

```
1  int a[10], b[10], c[10];
2
3  for (i = 0; i < 10; i++) {
4      a[i] = b[i] + c[i];
5  }
```

Código 1: *loop* DOALL

Exemplo de *loop* DOACROSS:

```
1  int i;
2  int fibo[10];
3
4  fibo[0] = fibo[1] = 1;
5  for(i = 2; i < 10; i++){
6    fibo[i] = fibo[i-1] + fibo[i-2];
7  }
```

Código 2: *loop* DOACROSS

[2]Mais sobre *loops*: https://en.wikipedia.org/wiki/Loop-level_parallelism#DOALL_parallelism.

Os *loops* do algoritmo BubbleSort são do tipo DOACROSS. Observe no código 3.

```
1  int bubbleSort(int * v, int n){
2    int i, j, tmp;
3    for(i = n; i >= 2; i--){
4      for (j = 0; j < i - 1; j++){
5        if(v[j] > v[j+1]){
6          tmp = v[j];
7          v[j] = v[j+1];
8          v[j+1] = tmp;
9        }
10     }
11   }
12 }
```

Código 3: BubbleSort

Altere o algoritmo BubbleSort para transformar o *loop* interno em DOALL, mantendo a complexidade do algoritmo original. Justifique que o novo algoritmo está correto.

Dica: você pode adicionar ou remover *loops* e modificar o *loop* externo, mas lembre-se de manter a complexidade do algoritmo original.