Universidade Federal de Minas Gerais

DCC002: Algoritmos e Estrutura de Dados 2

Professor: Flavio Vinicius Diniz de Figueiredo

Data: 18 de Abril de 2017

Nome Completo:

Prova 1: Análise de Complexidade e Revisão C

Questão 1 (4 pts): Assinale verdadeiro (V) ou falso (F) para cada afirmativa abaixo. Em todas as afirmativas, justifique sua resposta. Respostas sem justificativa não serão consideradas.

1. Sejam duas funções $f(n) = 7n^2 + 2\sqrt{n}$ e $g(n) = 6n^2 + 200n + 1$. É correto afirmar que um programa P1 com complexidade $f(n)$ é mais rápido que um programa P2, com complexidade $g(n)$ (assuma que n é maior do que uma entrada suficientemente grande n_0).

   Resposta:

   As duas funções são iguais do ponto de vista assintótico. Como ambas são $O(n^2)$ a afirmativa é **falsa**. Além disto, as constantes de $f(n)$ sempre são maiores do que aquelas de $g(n)$, até do ponto de vista não assintótico esperamos que $f(n)$ seja mais lento do que $g(n)$.

2. Um programa P executa uma função F1 com complexidade $f(n)$ em 90% de suas n iterações, e uma função F2 com complexidade $g(n)$ nas demais iterações. Qual a complexidade final O da função? Escreva a mesmo em termos de $f(n)$ e $g(n)$.

   Resposta:

   $$O(P) = O(0.9 * n * f(n)) + O(0.1 * n * g(n)) = O(n * max(f(n), g(n)))$$

3. Considere um programa P cuja função de complexidade é $f(n) = 3\log n$. É correto afirmar que esse programa tem complexidade $O(\log n)$, mas não tem complexidade $O(n \log n)$.

   Resposta:

   O programa é ao mesmo tempo $O(\log n)$ como também é $O(n \log n)$. Porém, $O(\log n)$ é um limite mais firme para o mesmo. Então a afirmativa é **falsa**.

4. Considere um programa P que faz uma série de operações de custo constante, chama uma função F1 com complexidade dada por $f(n)$ e depois chama uma função F2 com complexidade dada por $g(n)$, onde $g(n) = \frac{f(n)}{1000}$. Pode-se afirmar que o programa P é $O(f(n))$.

   Resposta:

   Sim. $g(n)$ é $O(f(n))$. O programa é:

   $$O(P) = O(f(n)) + O(g(n)) = O(max(f(n), g(n))) = O(f(n))$$

Questão 2 (7pts) Em Funes o Memorioso, Jorge Luis Borges descreve a vida de Funes, um uruguaio, que após um tombo, perdeu a capacidade de esquecer (Ficções, 1944) "Sabia as formas das nuvens austrais do amanhecer do trinta de abril de mil oitocentos e oitenta e dois e podia compará-las na lembrança aos veios de um livro encadernando em couro que vira somente uma vez e às linhas da espuma que um remo levantou no rio Negro às vésperas da batalha do Quebracho." Suponha que o passatempo de Funes fosse associar um número para a felicidade/tristeza que sentiu em cada um de seus dias. Quanto menor o número, mais infeliz foi o dia de Funes. Por exemplo, em 30/04/1882 seu índice de felicidade pode ter sido -5. Dado um vetor de números inteiros que representam os índices de felicidade de Funes (o primeiro elemento do vetor é o índice de felicidade para o primeiro dia e assim sucessivamente) retorne o valor acumulado de felicidade para o período contínuo de dias mais feliz de Funes.

O método tem a seguinte forma:

```
int somaMaxDiasFelicidade(int *felicidade, int n)
```

A resposta deve ser escrita em C. Indique e justifique a complexidade da sua função (respostas de complexidade $O(n^2)$ valem mais do que as de $O(n^3)$; respostas de complexidade $O(n)$ valem pontos extra).

Resposta1:

```
int somaMaxDiasFelicidade1(int *felicidade, int n) {
  //Quero testar todos os intervalos possíveis.
  //[0, 0]
  //[0, 1]
  //[0, 2]
  //...
  //[0, n-1]
  //[1, 1]
  //[1, 2]
  //...
  //[1, n-1]
  //...
  //...
  int i;
  int j;
  int k;
  int resposta = felicidade[0];
  int somaSubVetor = 0;
  for (i = 0; i < n; i++) {      //inicio do intervalo em i
    for (j = i; j < n; i++) {    //fim em j
      somaSubVetor = 0;
      for (k = i; k < j; k++) {  //soma dos elementos de [i, j)
        somaSubVetor += felicidade[k];
        if (somaSubVetor > resposta) { //é melhor do que a resposta?
          resposta = somaSubVetor;     //atualiza
        }
      }
    }
  }
  return resposta;
}
```

A função tem complexidade $O(n^3)$.

Resposta2:

```
int somaMaxDiasFelicidade2(int *felicidade, int n) {
  //Eu preciso do terceiro laço na solução anterior??
  //Não, posso computar a soma direto no segundo
  int i;
  int j;
  int resposta = felicidade[0];
  int somaSubVetor = 0;
  for (i = 0; i < n; i++) {          //inicio do intervalo, i
    somaSubVetor = 0;                //zero a soma do subVetor
    for (j = i; j < n; i++) {        //testo todas as possibilidades de [i, n)
      somaSubVetor += felicidade[j];
      if (somaSubVetor > resposta) { //é melhor? atualiza resposta.
        resposta = somaSubVetor;
      }
    }
  }
  return resposta;
}
```

A função tem complexidade $O(n^2)$.

Resposta3:

```
int somaMaxDiasFelicidade3(int *felicidade, int n) {
  int i;
  int resposta = felicidade[0];
  int somaAteAgora = felicidade[0];
  //O somaAteAgora vai acumulando a soma do vetor até ficar negativo.
  //Neste caso, zeramos a somaAteAgora
  //A intuição é que enquanto a somaAteAgora cresce, atualizamos o resultado
  //  Caso decresca, não atualizamos
  //  Caso zere ou fique negativo, os elementos após a melhor resposta não
  //  ajudam em nada, espero o próximo elemento positivo.
  for (i = 1; i < n; i++) {
    if (somaAteAgora + felicidade[i] > 0) {
      somaAteAgora = somaAteAgora + felicidade[i];
    } else {
      somaAteAgora = 0;
    }
    if (somaAteAgora > resposta) { //Atualiza a resposta caso seja uma boa
      resposta = somaAteAgora;
    }
  }
  return resposta;
}
```

A função tem complexidade $O(n)$.

Questão 3 (4pts) O produto escalar, também denominado produto interno, é o produto de dois vetores que resulta em um escalar. Por exemplo, o produto escalar entre dois vetores $\vec{u}$ e $\vec{v}$ é dado por:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n,$$

onde n é a dimensão dos vetores $\vec{u}$ e $\vec{v}$.

Uma outra operação comum entre vetores é o produto externo $\otimes$. O mesmo é equivalente à multiplicação matricial $\vec{u}\vec{v}^{\mathrm{T}}$. Por exemplo, para dois vetores $\vec{u}$ de tamanho 4 e $\vec{v}$ de tamanho 3, podemos definir o produto externo $\vec{u} \otimes \vec{v}$ como sendo:

$$\vec{u} \otimes \vec{v} = \vec{u}\vec{v}^{\mathrm{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}.$$

Com base nessas afirmativas, escreva na linguagem C:

(a) um TAD para armazenar a dimensão e os componentes de um vetor de números inteiros. Garanta que seu TAD defina structs para `vetor` e `matriz`. Você vai precisar dos dois. Para a prova, defina no TAD apenas as funções necessárias para as perguntas abaixo. No mais, indique a complexidade de cada função.

(b) funções para alocar vetores e matrizes:

```
vetor *alocaVetor(int n)
```

```
matriz *alocaMatriz(int nLinhas, int nColunas)
```

(c) uma função que, dados apontadores para dois vetores de mesma dimensão, retorna o produto escalar desses vetores. Sua função deverá ter a assinatura:

```
int produtoEscalar(vetor *v1, vetor *v2)
```

(d) uma função que, dados apontadores para dois vetores de mesma dimensão, returna o produto externo dos dois vetores. Sua função deverá ter a assinatura:

```
matriz *produtoExterno(vetor *v1, vetor *v2)
```

```c
#IFNDEF TAD_VEC_MAT_H
#DEFINE TAD_VEC_MAT_H

typedef struct {
  int n;
  int *dados;
} vetor;

typedef struct {
  int nLinhas;
  int nColunas;
  int **dados;
} matriz;

vetor *alocaVetor(int n);
matriz *alocaMatriz(int nLinhas, int nColunas);
int produtoEscalar(vetor *v1, vetor *v2);
matriz *produtoExterno(vetor *v1, vetor *v2);

#ENDIF
```

```c
vetor *alocaVetor(int n) {
  vetor *retorno = (vetor *) malloc(sizeof(vetor));
  if (retorno == NULL)
    exit(1);
  retorno->dados = (int *) malloc(n * sizeof(int));
  if (retorno->dados == NULL)
    exit(1);
  retorno->n = n;
  return retorno;
}

matriz *alocaMatriz(int nLinhas, int nColunas) {
  matriz *retorno = (matriz *) malloc(sizeof(matriz));
  if (retorno == NULL)
    exit(1);

  retorno->dados = (int **) malloc(nLinhas * sizeof(int));
  if (retorno->dados == NULL)
    exit(1);

  int i;
  for (i = 0; i < nLinhas; i++) {
    retorno->dados[i] = (int *) malloc(nColunas * sizeof(int));
    if (retorno->dados[i] == NULL)
      exit(1);
  }

  retorno->nLinhas = nLinhas;
  retorno->nColunas = nColunas;
  return retorno;
}

int produtoEscalar(vetor *v1, vetor *v2) {
  int resposta = 0;
  int i;
  for (i = 0; i < v1->n; i++)
    resposta += v1->dados[i] * v2->dados[i];
  return resposta;
}

matriz *produtoExterno(vetor *v1, vetor *v2) {
  matriz *resposta = alocaMatriz(v1->n, v2->n);
  int i;
  int j;
  for (i = 0; i < resposta->nLinhas; i++)
    for (j = 0; j < resposta->nColunas; j++)
      resposta->dados[i][j] = v1->dados[i] * v2->dados[j];
  return resposta;
}
```