UFMG/ICEx/DCC — Projeto e Análise de Algoritmos

# Lista de Exercícios 3

Pós-Graduação em Ciência da Computação — 1º Semestre de 2011

**Observações**:

1. Comece a fazer esta lista imediatamente. Você nunca terá tanto tempo para resolvê-lo quanto agora!
2. **Data de Entrega**: até 10 de junho, **às 9:30 horas**, ou antes. Após essa data e hora haverá uma penalização por atraso: $2^d$, onde $d$ é o número de dias de atraso.
3. Envie qualquer material referente a esta lista de exercícios para o endereço eletrônico `esub.para.loureiro@gmail.com` tendo como assunto `[PAA 2011/1 LE3: "seu nome completo"]` e como anexo um arquivo zip, descrito abaixo, com o nome `LE3_"SeuNomeCompleto".zip` onde o string `"SeuNomeCompleto"` é o seu nome completo sem espaços em branco.

   Exemplo para o aluno Zoroastro Felizardo e Sortudo:

   - Assunto: `[PAA 2011/1 LE3: Zoroastro Felizardo e Sortudo]`
   - Arquivo zip: `LE3_ZoroastroFelizardoESortudo.zip`
4. O seu programa deve ser executado em alguma máquina do ambiente computacional do Departamento de Ciência da Computação da UFMG, onde os monitores irão avaliá-lo. No arquivo `leiame.txt`, a ser incluído no arquivo zip, você deve dizer qual é o ambiente computacional para executar o seu TP bem como todas as instruções necessárias.
5. Linguagem do exercício de programação: C, C++ ou Java.
6. As questões, a seguir, tratam do projeto de algoritmos. CLSR é a referência do exercício no livro-texto.

---

**Observação**: As questões, a seguir, tratam do projeto de algoritmos. CLSR é a referência do exercício no livro-texto, segunda edição. As questões estão em inglês por terem sido copiadas da versão original.

## Questão 1 [CLRS, Ex 22.1-8, pg 531]

Suppose that instead of a linked list, each array entry $Adj[u]$ is a hash table containing the vertices $v$ for which $(u, v) \in E$. If all edge lookups are equally likely, what is the expected time to determine whether an edge is in the graph? What disadvantages does this scheme have? Suggest an alternate data structure for each edge list that solves these problems. Does your alternative have disadvantages compared to the hash table?

## Questão 2 [CLRS, Ex 22.2-7, pg 539]

The diameter of a tree $T = (V, E)$ is given by

$$\max_{u,v \in V} \delta(u, v),$$

that is, the diameter is the largest of all shortest-path distances in the tree. Give an efficient algorithm to compute the diameter of a tree, and analyze the running time of your algorithm.

## Questão 3 [CLRS, Ex 22.3-12, pg 549]

A directed graph $G = (V, E)$ is *singly connected* if $u \rightsquigarrow v$ implies that there is at most one simple path from $u$ to $v$ for all vertices $u, v \in V$. Give an efficient algorithm to determine whether or not a directed graph is singly connected.

**Questão 4** **[CLRS, Ex 22.5-1, pg 557]**

How can the number of strongly connected components of a graph change if a new edge is added?

**Questão 5** **[CLRS, Ex 22.5-3, pg 557]**

Professor Deaver claims that the algorithm for strongly connected components can be simplified by using the original (instead of the transpose) graph in the second depth-first search and scanning the vertices in order of increasing finishing times. Is the professor correct?

**Questão 6** **[CLRS, Ex 22.5-7, pg 557]**

A directed graph G = (V, E) is said to be semiconnected if, for all pairs of vertices $u, v \in V$, we have $u \rightsquigarrow v$ or $v \rightsquigarrow u$. Give an efficient algorithm to determine whether or not $G$ is semiconnected. Prove that your algorithm is correct, and analyze its running time.

---

Para o problema abaixo, escreva o algoritmo, faça a implementação, testes e apresente o custo de complexidade identificando a operação considerada relevante. Lembre-se que a linguagem de programação é C/C++. No caso de apresentar uma solução recursiva, discuta também e apresente a complexidade para o crescimento da pilha.

**Exercício de Programação 1: Estados globais de uma execução**

De todos os "objetos" matemáticos usados em Ciência da Computação, o grafo tem um papel de fundamental importância. É possível modelar vários dos problemas usando esse objeto.

Este exercício de programação é motivado por um cenário típico que ocorre em uma computação (execução) em um único elemento computacional ou em um conjunto de elementos computacionais. Sejam $n$ computações (execuções) representadas por $n$ *threads* em um único elemento computacional ou por $n$ processos, cada um executado em um elemento computacional distinto. Essas $n$ *threads* ou $n$ processos serão identificados apenas pela letra $p_i$, sendo $1 \leq i \leq n$. Essas $n$ tarefas podem trocar dados entre si através de "mensagens", ou seja, não existe uma memória compartilhada entre as diferentes execuções.

Uma atividade típica após a execução das $n$ tarefas é saber se uma determinada propriedade foi satisfeita ou não. Para isso, é possível construir um grafo de execução das $n$ atividades, percorrer esse grafo e avaliar em cada "estado" do sistema a propriedade de interesse.

Seja, por exemplo, uma computação envolvendo $p_1$ e $p_2$ como mostrado na figura 1. Nessa computação, os "momentos" (eventos) de interesse são representados por $e_i^k$, onde o subscrito $i$ representa o processo $i$ e o superscrito $k$ representa o $k$-ésimo evento de interesse em $p_i$. Um evento ocorre por uma mudança no estado da computação que é importante do ponto de vista de todo o sistema e, do ponto deste exercício, não é importante saber o que ocorreu efetivamente.

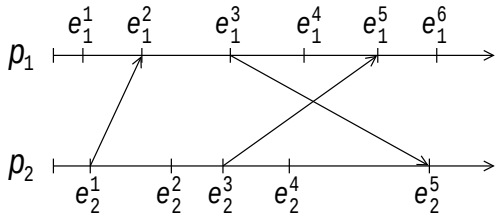


Figura 1: Computação envolvendo $p_1$ e $p_2$.

A questão passa a ser quais são as combinações de estados válidos que podem ter ocorrido na computação envolvendo $p_1$ e $p_2$. Para isso, podemos gerar um grafo que representa os possíveis estados válidos como mostrado na figura 2. Nessa figura, os vértices têm dois algarismos, sendo que o primeiro diz respeito ao número do evento do processo $p_1$ e o segundo ao número do evento do processo $p_2$. Assim, o vértice 00 quer dizer que é possível ter o nosso sistema de interesse em um determinado momento com $p_1$ em $e_1^0$ e $p_2$ em $e_2^0$, que são os eventos iniciais e não estão explicitados na figura. A partir desse estado, a computação pode ir tanto para 10 ou 01. No primeiro caso, temos $p_1$ em $e_1^1$ e $p_2$ em $e_2^0$. Note que não é possível ter um vértice que representa a computação 20. Isso

implicaria que teríamos $p_1$ em $e_1^2$ e $p_2$ em $e_2^0$, ou seja, $p_1$ teria alcançado $e_1^2$ enquanto $p_2$ ainda estaria em $e_2^0$. Mas isso não é possível já que a ocorrência de $e_1^2$ depende da ocorrência de $e_2^1$ que ocorre depois de $e_2^0$.

Note que esse grafo é dirigido mas neste caso estamos usando uma representação especial que se chama reticulado (*lattice*). O reticulado é a transformação de um grafo dirigido em um grafo não-dirigido. Para isso, representa-se o grafo dirigido com todas as arestas desenhadas de baixo para cima. O vértice inicial ou de entrada desse grafo é o vértice mais embaixo (neste caso, o vértice 00). Assim, todo caminhamento sempre ocorre "para cima". O grafo não-dirigido é obtido eliminando-se o sentido das arestas e o caminhando continua sendo feito "para cima".

Na figura 3, a linha mais grossa mostra uma possível computação desse sistema envolvendo $p_1$ e $p_2$, começando no vértice 00 e terminando no vértice 65. De forma mais precisa, o caminhamento que representa essa computação pode ser expresso como a sequência de vértices $00, 01, 11, 21, 31, 32, 42, 43, 44, 54, 64, 65$.

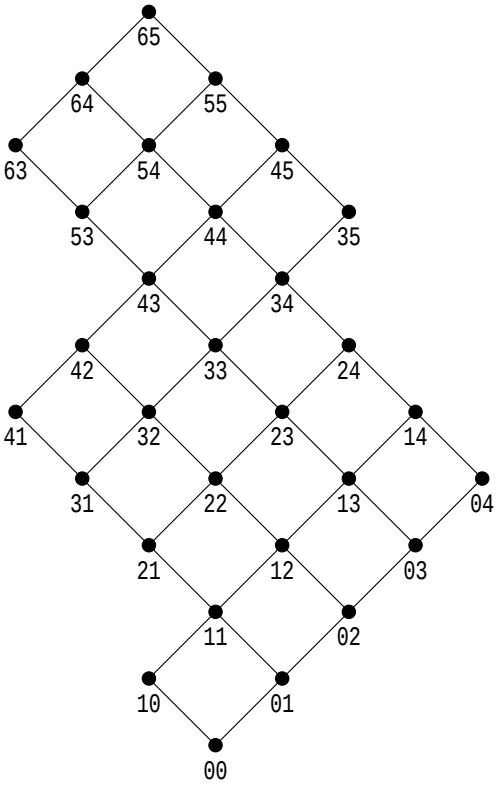


Figura 2: Grafo da computação envolvendo $p_1$ e $p_2$.

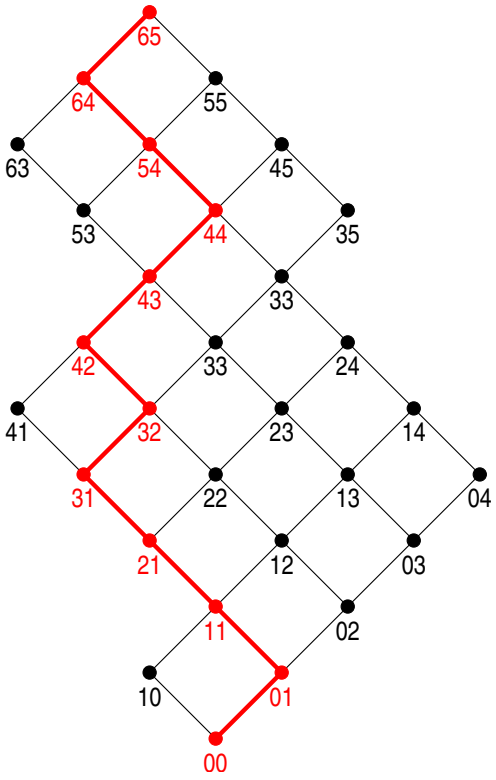


Figura 3: Possível execução do grafo da computação envolvendo $p_1$ e $p_2$.

**Pede-se:** gerar o grafo (reticulado) das possíveis computações de $n$ atividades, sendo $2 \leq n \leq 4$, e avaliar propriedades de interesse nesse grafo. No caso da propriedade ser satisfeita, deve-se dizer se a propriedade ocorreu com certeza ou se pode ter ocorrido. A propriedade deve ser expressa como uma expressão proposicional. Uma expressão proposicional, ou condição, é uma expressão que possui variáveis que se transforma numa proposição, i.e., possui um valor lógico verdadeiro ou falso, quando se substituem essas variáveis por valores.

**Suposições:** para resolver este trabalho, faça as seguintes suposições:

1. A expressão proposicional é formada por variáveis apenas do tipo inteiro;
2. O evento $e_i^k$ em $p_i$ será identificado no arquivo de entrada pelo número inteiro $k$;
3. A variável $v_i^k$ em $p_i$ será identificada no arquivo de entrada pelo número inteiro $k$;
4. Uma expressão proposicional pode ter os seguintes operadores (representados entre colchetes pelo símbolo a ser fornecido no arquivo de entrada):

- Operadores lógicos: $\neg$ (negação) [`~`]; $\wedge$ (disjunção) [`|`]; $\vee$ (conjunção) [`.`]; $\rightarrow$ (condicional) [`I`]; $\leftrightarrow$ (bi-condicional) [`B`];
- Operadores aritméticos: adição [`+`]; subtração [`-`]; multiplicação [`*`]; divisão [`/`];
- Operadores relacionais: "maior que" [`>`]; "menor que" [`<`] e "igualdade" [`=`].

5. Expressões podem ter parênteses e a prioridade é a mesma de uma expressão proposicional;

6. Como foi explicado acima, existem eventos em diferentes atividades que estão relacionados entre si. Por exemplo, na figura 2, os eventos $e_2^1$ e $e_1^2$ estão relacionados entre si, sendo que a ocorrência do primeiro implica na ocorrência do segundo. No arquivo de entrada, na linha de descrição do evento, isso será codificado com uma referência da seguinte forma: $\langle e_i^k$[`O`|`D`]`P`$p_j\rangle$, onde $e_i^k$ indica o número do $k$-ésimo evento em $p_i$ (este é um número sequencial que começa em 0), `O` caso seja um evento que está relacionado com outro que tem origem em $p_j$ e `D` caso seja um evento que está relacionado com outro que tem destino em $p_j$. Essa codificação pode ser usada a partir da terceira linha do arquivo de entrada como discutido abaixo;

7. Uma variável $v_i^k$ em $p_i$ será identificada em uma expressão proposicional como $\langle$`V`$v_i^k$`P`$p_i\rangle$, onde $v_i^k$ indica o número da $k$-ésima variável em $p_i$.

**Entrada:** o formato de cada linha do arquivo de entrada está descrito abaixo:

| **Formato de cada linha do arquivo de entrada** | `Exemplo` |
|---|---|
| Nº de atividades | `2` |
| Nº de variáveis associadas a $p_1 \dots p_n$ | `2 3` |
| Processo $p_i$, evento $e_i^k$, valores das variáveis $v_i^1 \dots$ | `1 0 0 0` |
| | `1 1 2 0` |
| | `1 2D2P2 3 1` |
| | `1 3O5P2 5 4` |
| | `1 4 7 9` |
| | `1 5D3P2 11 16` |
| | `1 6 13 25` |
| | `2 0 0 0 0` |
| | `2 1O1P1 1 -1 3` |
| | `2 2 2 3 5` |
| | `2 3O5P1 3 -5 7` |
| | `2 4 4 7 9` |
| | `2 5D3P1 5 -9 11` |
| $\langle$*Id do predicado*$\rangle$: $\langle$*predicado*$\rangle$ | `P1:  V1P1 > V3P2` |