UFMG/ICEx/DCC — Projeto e Análise de Algoritmos

3ª Lista de Exercícios

Pós-Graduação em Ciência da Computação — 1º Semestre de 2010

Observações:

- Assuma que a estrutura de dados usada para representar um grafo é a lista de adjacência. Se você optar por outra estrutura, justifique a sua escolha.

Questão 1

Temos três recipientes cujos volumes são 10, 7 e 4 litros, respectivamente. Os recipientes de 7 e 4 litros começam cheios de água, mas o de 10 litros está inicialmente vazio. É permitido apenas um tipo de operação: despejar o conteúdo de um recipiente em outro, parando somente quando o recipiente de origem estiver vazio ou quando o recipiente de destino estiver cheio. Queremos saber se existe uma sequência de despejos que deixe exatamente 2 litros no recipiente de 7 litros ou no de 4 litros.

(a) Modele como um problema em grafos: forneça uma definição precisa do grafo envolvido e formule a questão específica sobre esse grafo que precisa ser respondida.

(b) Qual algoritmo deve ser aplicado para resolver este problema?

(c) Proponha uma "metodologia genérica" (princípios) que podem ser aplicados e seguidos para resolver este tipo de problema.

Questão 2

Para cada vértice v em um grafo não-dirigido, seja `GrausVizinhos[v]` a soma dos graus dos vizinhos de v. Mostre como computar o vetor inteiro de valores de `GrausVizinhos` em tempo linear na quantidade de vértices e arestas.

Questão 3

Apresente pelo menos duas propriedades que um grafo não-dirigido deve ter para ser bipartido. Prove essas propriedades.

Questão 4

Apresente um algoritmo linear na quantidade de vértices e arestas para determinar se um grafo não-dirigido é bipartido.

Questão 5

Apresente um algoritmo eficiente que receba como entrada um grafo dirigido acíclico $G = (V, E)$ e dois vértices $s, t \in V$ e compute o número de caminhos diferentes de s a t em G.

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: 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 (i.e., ocorreu, tornou-se verdadeira) ou não durante a computação. Para isso, é possível construir, após o término da computação, um grafo das possíveis execuções das n atividades (incluindo naturalmente o sub-grafo que representa a computação que efetivamente ocorreu mas que, em geral, não é possível saber qual é), percorrer esse grafo e avaliar em cada "estado" do sistema a propriedade de interesse. Dependendo da estrutura do grafo, é possível afirmar se a propriedade ocorreu com certeza ou afirmar que pode ter ocorrido.

Seja, por exemplo, uma computação envolvendo p_1 e p_2 como mostrado na figura 1. As linhas horizontais representam a linha do tempo. 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. É exatamente quando ocorre um evento que devemos avaliar se uma propriedade é satisfeita ou não. Uma seta de um processo para outro indica uma relação de dependência causal entre esses eventos, ou seja, a ocorrência de um evento em um processo (origem da seta) causa a ocorrência de um evento em outro processo (término da seta).

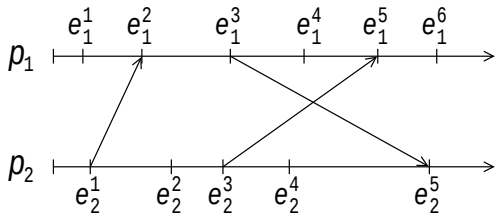

Figura 1: Computação envolvendo p_1 e p_2.

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 significa 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. No segundo caso, temos p_1 em e_1^0 e p_2 em e_2^1. 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 $00, 01, 11, 21, 31, 32, 42, 43, 44, 54, 64, 65$.

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) [`~`], $\vee$ (disjunção) [`|`], $\wedge$ (conjunção) [`.`], $\rightarrow$ (condicional) [`I`], $\leftrightarrow$ (bi-condicional) [`B`];
   - Operadores aritméticos: adição [`+`], subtração [`-`], multiplicação [`*`] e 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 $p_1, \ldots, p_n$	`2 3`
Processo p_i, evento e_i^k, valores das variáveis $v_i^1 \ldots$	`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`

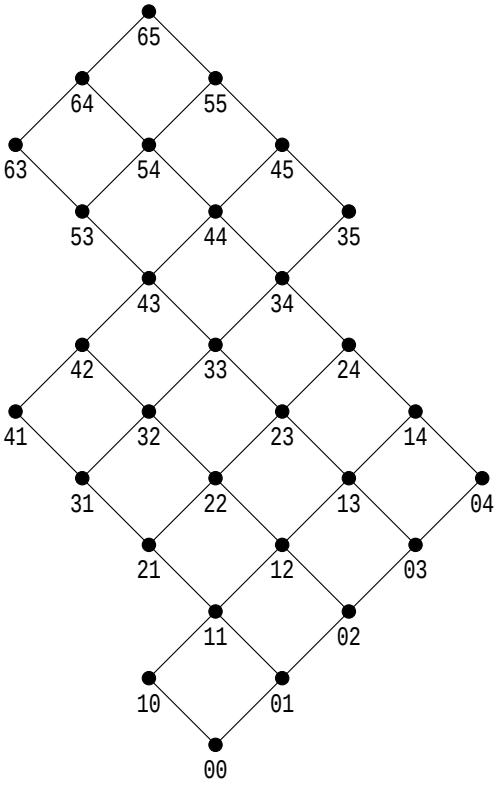

Figura 2: Grafo da computação envolvendo p_1 e p_2.

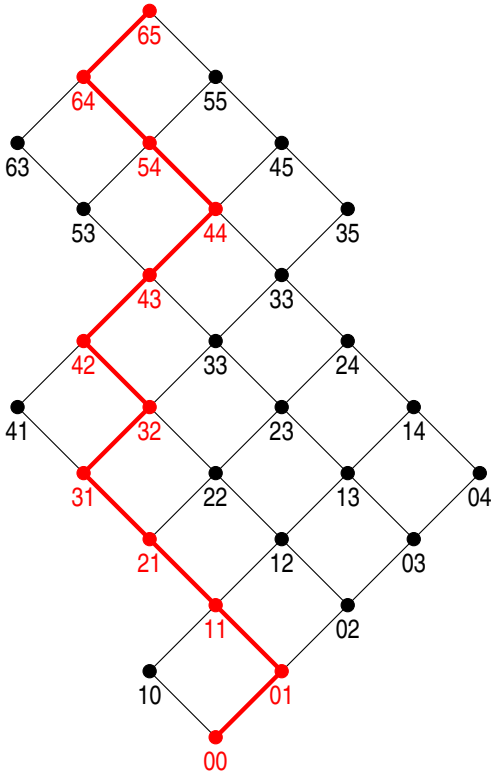

Figura 3: Possível execução do grafo da computação envolvendo p_1 e p_2.