MAC323 EXERCÍCIO-PROGRAMA 2

PONTEIROS DANÇANTES E COLORAÇÃO DE GRAFOS

Y. KOHAYAKAWA

Data de entrega: 22/4/2008

Introdução. Este EP supõe que você está familiarizado com o *Problema da Cobertura Exata* (PCE) e sua generalização, o *Problema da Cobertura Generalizado* (PCG). Ademais, você precisa conhecer o *Algoritmo X* de Knuth, implementado com *ponteiros dançantes*, que resolve o PCE e o PCG. A referência principal é

- ▷ D. E. Knuth, Dancing links, *Millennial Perspectives in Computer Science*, J. Davies, B. Roscoe, and J. Woodcock, editors, Palgrave, Houndmills, 2000, pp. 187-214. Versão preliminar disponível em `http://arxiv.org/abs/cs/0011047`

Neste EP, você

(*i*) deverá implementar o Algoritmo X de Knuth, com ponteiros dançantes, e

(*ii*) deverá usar sua implementação de X para resolver o problema da coloração de grafos: dados um grafo G e um inteiro k, decidir se G é *k-colorível*, isto é, se G tem uma coloração própria de seus vértices com k cores.

Embora você não precisará de mais do que a definição de coloração de grafos, para ler um pouco sobre isso, você pode consultar `http://en.wikipedia.org/wiki/Graph_coloring` (estamos interessados em coloração de vértices (*vertex colourings*)).

Ao terminar este EP, você terá um programa que será capaz de decidir se um dado grafo é k-colorível, para qualquer k dado. Para testar a eficiência de seu programa, você deverá fazer experimentos computacionais para tentar determinar um "ponto crítico" na evolução de um grafo: "o tempo τ da evolução em que o grafo deixa de ser 3-colorível" (isso é explicado a seguir).

As três partes do EP. A sua tarefa tem três fases: (A) implementar o assim chamado *Algoritmo DLX*, o Algoritmo X com ponteiros dançantes, (B) implementar um algoritmo que, dado um grafo G e um inteiro k, produz uma matriz, digamos $M(G, k)$, que, quando dado como entrada para o PCG, há uma cobertura para $M(G, k)$ como o PCG exige se e só se G é k-colorível, (C) projetar e implementar experimentos para estudar τ (mencionado brevemente acima).

Você deverá escrever três programas: `ep2a.c`, `ep2b.c`, e `ep2c.c`.

Fase A. Seu programa `ep2a.c` deve ser uma implementação o Algoritmo DLX. As colunas da matriz de entrada M devem ter nomes simbólicos: *strings* com no máximo, digamos, *max_col_name_len* caracteres (por exemplo, 50 caracteres). Lembre-se que há colunas de dois tipos: *primárias* e *secundárias*. As linhas da matriz M serão dadas pelas colunas nas quais elas têm entrada 1 (isto é exatamente como no artigo de Knuth).

Date: Versão de 31 de março de 2008 (21:18).

Para um usuário dar uma matriz M como entrada para o seu programa, ele colocará na primeira linha da entrada a lista dos nomes das colunas primárias e secundárias, separadas pelo caracter `'|'`. Por exemplo, como discutido em sala, para resolver o problema das 4 rainhas, o usuário poderá fornecer como entrada para o seu programa a entrada

```
R0 R1 R2 R3 F0 F1 F2 F3 | A0 A1 A2 A3 A4 A5 A6 B0 B1 B2 B3 B4 B5 B6
R0  F0  A0  B3
R0  F1  A1  B4
R0  F2  A2  B5
R0  F3  A3  B6
R1  F0  A1  B2
R1  F1  A2  B3
R1  F2  A3  B4
R1  F3  A4  B5
R2  F0  A2  B1
R2  F1  A3  B2
R2  F2  A4  B3
R2  F3  A5  B4
R3  F0  A3  B0
R3  F1  A4  B1
R3  F2  A5  B2
R3  F3  A6  B3
```

A saída de seu programa deverá ser o número de soluções para a matriz dada (no caso da entrada acima, há duas soluções). Seu programa também deve ser capaz de imprimir as soluções encontradas. Por exemplo, para a entrada acima, a saída poderia ser

```
1:
 R0 F1 A1 B4
 R1 F3 A4 B5
 R2 F0 A2 B1
 R3 F2 A5 B2
2:
 R0 F2 A2 B5
 R1 F0 A1 B2
 R2 F3 A5 B4
 R3 F1 A4 B1
```

Fase B. O seu programa `ep2b.c` deve receber como entrada um grafo G e um inteiro $k \geq 1$ e deve ter como saída a matriz $M(G, k)$, como discutido acima. A entrada terá o formato ilustrado no exemplo a seguir. Suponha que G é o circuito com 5 vértices e que $k = 3$. Assim, G tem, por exemplo, vértices $a, b, \ldots, e$ e arestas $\{a, b\}, \{b, c\}, \ldots, \{a, e\}$. A entrada para o seu `ep2b.c` poderia então ser

```
3 5
a: b e
b: a c
c: b d
d: e c
e: a d
```

São dados inicialmente os valores de k e o número de vértices no grafo. A seguir, são listados os vértices e seus vizinhos (vértices adjacentes àqueles vértices). Suporemos que os vértices dos grafos são identificados por *strings* (de comprimento no máximo, digamos, *max_v_name_len*).

Note que, com os programas `ep2a.c` e `ep2b.c` implementados, você já terá um sistema que, dados um grafo G e um inteiro k, decide se G é k-colorível. Basta você executar

```
prompt > ep2b < entrada.in | ep2a
```

Fase C. No EP1, você considerou *evolução de grafos*: você começou com um grafo vazio e foi adicionando arestas ao acaso, de forma que o grafo foi "crescendo" de acordo com um processo aleatório. O tópico de interesse foi quando o grafo ficou conexo. Isso basicamente acontece quando o número de arestas geradas é algo como $N(\log N)/2$, e você deve ter confirmado isso experimentalmente.

Nesse EP, podemos fazer algo semalhante ao que foi feito no EP1, focando, entretanto, nossa atenção no momento em que o grafo deixa de ser 3-colorível; isto é, podemos determinar o menor τ com a propriedade de que com $\tau - 1$ arestas o grafo é 3-colorível, mas com τ arestas o grafo já não é. Há porém uma dificuldade: decidir conexidade é muito mais fácil do que decidir se um dado grafo é 3-colorível (sabe-se que este último problema é NP-completo, de forma que se acredita que não há algoritmos eficientes para ele). Assim, nossos experimentos devem minimizar o número de vezes que testamos 3-colorabilidade. Faremos assim uma simplificação.

Seu programa `ep2c.c` deve receber argumentos `-nN`, `-mM`, `-sS`, e `-rR`, onde N, M, S, e R são números naturais, interpretados da seguinte forma. Devem ser gerados grafos aleatórios com N vértices e M arestas: isto é, você deve gerar M pares da forma $\{x, y\}$, com $0 \leq x, y < N$, como no EP1, sem reposição, e você deve considerar o grafo com N vértices e M arestas assim obtido. Você deve então decidir se esse grafo é 3-colorível ou não, usando os algoritmos implementados no `ep2a.c` e no `ep2b.c`. Este experimento deve ser repetido R vezes, e a saída de seu programa deve ser o número de vezes que o grafo gerado foi 3-colorível. Para o processo de geração das arestas aleatórias, você deve usar S como a semente de seu gerador de números aleatórios.

Com seu programa `ep2c.c` pronto, você poderá encontrar, para cada N, o valor de $\tau = \tau(N)$ em torno do qual ocorre a mudança: para valores de M consideravelmente menores que τ, a maioria dos grafos gerados é 3-colorível, enquanto que, para valores de M consideravelmente maiores que τ, a proporção de tais grafos é pequena. O ideal seria você conseguir estimar $\tau = \tau(N)$ para valores grandes de N (por exemplo, $N = 100$, 200 ou 1000). Sabe-se que, para $N \to \infty$, o valor de $\tau = \tau(N)$ é basicamente da forma cN para alguma constante $c > 0$, cujo valor é ainda desconhecido. Em seu relatório, você deve dizer qual é sua estimativa para c, e deve dizer os experimentos que você executou para chegar a esse valor (dê todos os detalhes, de forma que o monitor poderá refazer seus experimentos).

Alguns detalhes. O seu programa `ep2a.c` deve aceitar alguns argumentos na linha de comando. Sem argumento nenhum, seu programa deve simplesmente imprimir o número total de soluções que a matriz de entrada admite. Se o usuário especificar `-sM` (onde M é um inteiro positivo), seu programa deve imprimir também cada M-ésima solução, isto é, as j-ésimas soluções ($j = 0, M, 2M, \ldots$). Com o argumento `-d`, seu programa deve simplesmente decidir se há ou não uma solução (não é necessário contar o número de soluções). A opção `-D` deve fazer o mesmo, mas também deve imprimir uma solução.

Finalmente, `ep2a.c` deve também aceitar o argumento `-tT` (onde T é um natural). Com esse argumento, seu programa deve considerar apenas soluções com no máximo T linhas. Essa opção não é tão útil para o problema da coloração de grafos, mas é útil em outra aplicações.

Observações

1. *Este EP é estritamente individual.* Programas semelhantes receberão nota 0.
2. Seja cuidadoso com sua programação (correção, documentação, apresentação, clareza do código, etc), dando especial atenção a suas estruturas de dados. A correção será feita levando isso em conta.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue seu EP através do sistema Paca.
5. Não deixe de incluir em seu código um *relatório* para discutir seu EP: discuta as estruturas de dados usadas, os algoritmos usados, etc. *Se você escrever claramente como funciona seu EP, o monitor terá pouca dificuldade em corrigi-lo, e assim você terá uma nota mais alta.* (Se o monitor sofrer para entender seu código, você pode imaginar o humor dele ao atribuir sua nota.)
6. Lembre que o monitor colocou no fórum recomendações sobre os EPs. Não deixe de segui-las: ele é quem decide sua nota!

Observação final. Enviem dúvidas para a lista de discussão da disciplina.

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–090 São Paulo, SP

Endereço Eletrônico: `yoshi@ime.usp.br`