MAC323 EXERCÍCIO-PROGRAMA 2

SATISFAÇA-ME

Y. KOHAYAKAWA

Data de entrega: 23/4/2012 (23:55)

Introdução. Neste EP, você escreverá um programa para resolver o *k-SAT*, o problema da satisfatibilidade para *k-CNF*s, fórmulas booleanas em forma normal conjuntiva com cláusulas com k literais.

Este EP supõe que você está familiarizado com o algoritmo X e sua implementação com ponteiros dançantes (DLX). A fonte principal para este material é o artigo de Knuth (2000), *Dancing links*, `arXiv:cs/0011047`. Você deverá codificar as instâncias de k-SAT como instâncias de um problema de cobertura generalizado adequado e depois resolver estas condificações usando o algoritmo X (adequadamente adaptado).

Descrição do problema. O problema principal que queremos resolver é o k-SAT:

Problema k-SAT
Instância: uma k-CNF ϕ
Pergunta: ϕ é satisfatível?

Uma k-CNF ϕ é simplesmente uma coleção de *cláusulas* $c_1, \ldots, c_m$, com cada c_i um conjunto de k *literais*, isto é, *variáveis booleanas*, que supomos vir de um conjunto U, e suas *negações*. Por exemplo, seja ϕ a 3-CNF com as 5 cláusulas $c_1 = \{\neg x_1, x_2, x_3\}$, $c_2 = \{x_1, \neg x_2, x_4\}$, $c_3 = \{x_1, \neg x_3, x_4\}$ e $c_4 = \{\neg x_1, \neg x_2, x_3\}$ $c_5 = \{\neg x_1, x_2, \neg x_3\}$ (aqui, $U = \{x_1, x_2, x_3, x_4\}$). Interpretamos ϕ como a fórmula booleana

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \\ \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3). \qquad (1)$$

A pergunta é então se é possível encontrar uma *atribuição de valores V e F às variáveis em ϕ*, ou simplesmente uma *atribuição*,

$$\nu \colon U \to \{V, F\} \qquad (2)$$

que *satisfaça* (1). Uma tal atribuição é a seguinte: $\nu(x_1) = \nu(x_2) = \nu(x_3) = V$ e $\nu(x_4) = F$ (das $2^4 = 16$ possíveis, 7 delas satisfazem ϕ). Assim, ϕ é uma instância *positiva* para k-SAT.

Seu programa. Em sua versão mais simples, seu programa deve receber uma k-CNF ϕ e deve ter como saída 0 caso ϕ não seja satisfatível e 1 caso ela seja. O usuário deve também poder pedir para seu programa imprimir o *número* de atribuições que satisfazem ϕ e também deve poder pedir que as atribuições que satisfazem ϕ sejam impressas. A atribuição dada no exemplo acima deve ser codificada na saída como `1110`.

Versão de 3 de abril de 2012, 08:55.

O formato da entrada é exemplificado as seguir. Para codificar a 3-CNF ϕ do exemplo acima, o usuário fornecerá a entrada

```
3 4 5
-1 2 3
1 -2 4
1 -3 4
-1 -2 3
-1 2 -3
```

A primeira linha declara que se trata de uma instância para o 3-SAT, com 4 variáveis e 5 cláusulas. Seguem as cláusulas, com i representando a variável x_i e $-i$ representando $\neg x_i$. Seu programa deve receber a entrada no `stdin` e a saída deve ser enviada para o `stdout`.

Opções. Sem nenhum argumento, seu programa deve ter como saída 1 ou 0, indicando se a k-CNF de entrada é satisfatível ou não. Com a opção de linha de comando `-n`, seu program deve imprimir o número de atribuições que satisfazem a entrada. Ademais, com `-N`, seu programa deve imprimir o número e *todas* as atribuições que satisfazem a entrada, uma por linha, em ordem lexicográfica.

Seguem alguns exemplos de execução, supondo que `f.in` contém a instância ϕ acima:

```
$ ep2 < f.in
1
$ ep2 -n < f.in
7
$ ep2 -N < f.in
7
0000
0001
0011
0101
0111
1110
1111
$
```

Redução a um problema de cobertura. Seu programa *deve* reduzir k-SAT a um problema de cobertura, como visto em sala. Mais precisamente, você deverá conceber um problema de cobertura genralizado (PCG) adequado para codificar as instâncias de k-SAT. Você *deve* então implementar uma variante adequada o algoritmo X para resolver seu problema de cobertura, e você *deve* implementar seu algoritmo com listas ligadas ortogonais e ponteiros dançantes. Não deixe de estudar o programa DANCE, de Knuth:

`http://www-cs-faculty.stanford.edu/~uno/programs.html`

Em seu relatório, não deixe de explicar como é seu PCG e como é sua variante do algoritmo X.

Instâncias. Seu programa será testado com instâncias pequenas, cuidadosamente escolhidas, e também com instâncias grandes, em geral geradas de forma aleatória.

Fixe k, n e m. Podemos gerar instâncias aleatórias ϕ de k-SAT com n variáveis e m cláusulas escolhendo m das $2^k\binom{n}{k}$ possíveis cláusulas uniformemente a acaso, com reposição. Acredita-se que, para todo k, existe uma constante c_k com a seguinte propriedade: *Seja $\varepsilon > 0$ uma constante e suponha que $n \to \infty$. Se $m \geq (c_k + \varepsilon)n$, então ϕ é quase-certamente não-satisfatível, enquanto que se $m \leq (c_k - \varepsilon)n$, então ϕ é quase-certamente satisfatível.* Você deverá usar seu programa para estimar esta constante c_k (supondo que ela existe). Faça isso para $2 \leq k \leq 5$ e para alguns valores maiores de k (como 10 e outros).

Verifique o quão rapidamente seu programa consegue resolver k-SAT para entradas ϕ aleatórias, prestando atenção especialmente nos casos em que m está em torno de $c_k n$.

Uma opção para depuração (e correção). Com a opção de linha de comando `-C` seu EP deverá se comportar de maneira *especial*, aceitando e processando uma entrada diferente da proposta pelo enunciado. Mais especificamente, seu EP receberá um problema da cobertura exata e decidirá se ele possui solução, imprimindo-a se for o caso. Esse modo será útil para você testar o algoritmo DLX separadamente (e também ajudará a correção do EP). É obrigatória a implementação dessa opção.

A entrada representará o problema de cobrir o conjunto $[n] = \{1, \ldots, n\}$ usando uma subcoleção de uma coleção de m subconjuntos de $[n]$. A entrada será composta por dois inteiros n e m, seguidos de m linhas contendo inteiros pertencentes a n em ordem crescente — cada uma dessas linhas corresponde a um subconjunto de $[n]$.

A saída deve ser composta por um inteiro s indicando o número de soluções para o problema, seguido de s linhas, que representam tais soluções. Cada linha é composta por números entre 1 e m em ordem crescente, indicando os subconjuntos compõem a solução. As soluções podem estar em qualquer ordem.

A seguir alguns exemplos de entradas e saídas esperadas para esse modo:

Entrada 1:

```
5 4
1 4 5
2 3
2
3
```

Saida 1:

```
2
1 2
1 3 4
```

Entrada 2:

```
3 2
1 2
2 3
```

Saída 2:

```
0
```

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 levará isso em conta.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue seu EP no 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, sua nota será baixa.)

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`