MAC0239: Exercício-Programa 1

Versão 2

4 de setembro de 2017

1 Introdução

Mastermind é um jogo para dois jogadores onde um deles assume o papel de criador do código (*codemaker*) e o outro tentará decifrá-lo (*codebreaker*). O código é formador colocando pinos de uma das 6 em cores em cada um dos 4 espaços apropriados. Depois de criado o código, o *codebreaker* tem um número pré-determinado de tentativas para adivinhar o código gerado. A cada tentativa, o *codemaker* indicará por meio de pinos o quão próximo da resposta o chute do *codebreaker* está: para cada pino da cor correta na posição certa, um pino de cor preta de feedback é dado, e para cada pino de uma cor certa, mas em um posição incorreta é fornecido um pino de cor branca.

Mais informações sobre o jogo podem ser vistas na página da Wikipedia[1], além disso há uma versão do jogo para navegador em: `http://www.archimedes-lab.org/mastermind.html`.

[1] `https://en.wikipedia.org/wiki/Mastermind_(board_game)`

2 Modificações ao Mastermind Original

Neste exercício-programa propomos uma versão simplificada do Mastermind, no que diz respeito à natureza do feedback: em vez do que é dado no jogo original, cada posição do feedback indica se a cor naquela posição está correta ou não. Além disso, em vez trabalharmos com somente 6 cores e 4 espaços, vamos usar um número c de cores e um número n de espaços que serão fornecidos como argumento para o programa.

3 Modelagem

Para facilitar a modelagem (e simplificar o código), vamos indicar os espaços por naturais de 0 até $n-1$ e as cores por naturais de 0 até $c-1$. Assim, podemos usar variáveis proposicionais da forma x_{ij}, onde $v(x_{ij}) = 1$ se o pino no espaço i é da cor j.

4 Formato de Entrada

A maioria dos resolvedores SAT aceita entrada no formato "DIMACS CNF", que é um formato de texto simples. Toda linha iniciada por "c"é um comentário. A primeira linha não-comentário deve ser da forma:

p cnf NÚMERO_DE_VARIÀVEIS NÚMERO_DE_CLÁUSULAS

Cada linha não comentada após esta define uma cláusula. Cada uma dessas linha é uma lista de inteiros separados por espaço em branco. Um valor positivo indica que o literal correspondente à variáve (assim, 4 indica o literal p_4), e u valor negativo indica a negação desta variável (assim, -5 indica o literal $\neg p_5$). Cada linha deve terminar com o número 0 seguido ou não de espaço.

Assim, a fórmula

$$(p_1 \vee \neg p_5 \vee p_4) \wedge (\neg p_1 \vee p_5 \vee p_3 \vee p_4) \wedge (\neg p_3 \vee \neg p_4)$$

pode ser expressa com o seguinte arquivo `exemplo.cnf`

```
c Esse é um comentário
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

A linha iniciada por "p cnf" indica que esse é um problema SAT na forma normal clausal/conjuntiva, com 5 variáveis e 3 cláusulas.

Mais informações sobre isso no endereço `http://www.satcompetition.org/2004/format-solvers2004.html` .

5 Código

5.1 Versão em Python

Junto com este enunciado é fornecido o código do SAT solver Lingeling, vocês devem compilá-lo (no Linux, usando o comando `make`) e copiar o executável `lingeling` para a mesma pasta do arquivo `mastermind.py` fornecido na pasta do EP.

No arquivo `mastermind.py` há uma simulação do jogo Mastermind, neste código vocês devem alterar **somente** as funções `convert_feedback` e `_strategy_new`. A função `convert_feedback` traduz o feedback em fórmulas para o SAT solver. Essa função representa o que o *codebreaker* "aprende"com o feedback dado pelo *codemaker*. Caso todas as atribuições estejam corretas, ela termina devolvendo `True`, senão executa a função de estratégia dada (observe os comentários nas linhas 55-58 do arquivo `mastermind.py`).

Na função `_strategy_new`, você deve implementar uma estratégia que gere cláusulas de acordo com o feedback. Para servir de exemplo, o exercício programa vem com duas funções de estratégia bastante simples implementadas.

A função `_strategy_simple` adiciona uma cláusula dizendo que algum dos espaços está com a cor incorreta, isto é, se a tentativa foi: $(x_{1c_1}, x_{2c_2}, x_{3c_1}, x_{4c_2})$, produz a cláusula $\neg x_{1c_1} \vee \neg x_{2c_2} \vee \neg x_{3c_1} \vee \neg x_{4c_2}$.

Já a função `_strategy_full` adiciona para cara atribuição uma cláusula unitária indicando se foi correta ou falsa.

O script `mastermind.py` recebe 3 parâmetros, nesta ordem: número de espaços, número de cores e número de tentativas. Assim, a seguinte chamada: `python3 mastermind.py 4 6 10`, indica uma rodada do Mastermind com 4 espaços, 6 cores e 10 tentativas.

O programa imprime o código gerador pelo *codemaker* e cada uma das tentativas feitas pelo *codebreaker*. No final, ele imprime quem foi o vencedor.

5.2 Versão em Java

Para compilar o código Java basta executar os seguintes comandos na mesma pasta onde extrair os arquivos do EP:

```
javac br/usp/ime/liamf/*.java
jar cvfe mastermind.jar br.usp.ime.liamf.Main br/
```

Junto com este enunciado é fornecido o código do SAT solver Lingeling, vocês devem compilá-lo (no Linux, usando o comando `make`) e copiar o executável `lingeling` para a mesma pasta do arquivo `mastermind.jar` produzido.

O código Java está dividido em vários arquivos, mas vocês devem alterar **apenas** as funções `convertFeedback` e `strategyNew` no arquivo `CodeBreaker.java`.

A função `convertFeedback` traduz o feedback em fórmulas para o SAT solver. Essa função representa o que o *codebreaker* "aprende"com o feedback

dado pelo *codemaker*. Caso todas as atribuições estejam corretas, ela termina devolvendo `True`, senão executa a função de estratégia dada (observe os comentários nas linhas 67-71 do arquivo `CodeBreaker.java`).

Na função `strategyNew`, você deve implementar uma estratégia que gere clásulas de acordo com o feedback. Para servir de exemplo, o exercício programa vem com duas funções de estratégia bastante simples implementadas.

A função `strategySimple` adiciona uma cláusula dizendo que algum dos espaços está com a cor incorreta, isto é, se a tentativa foi: $(x_{1c_1}, x_{2c_2}, x_{3c_1}, x_{4c_2})$, produz a cláusula $\neg x_{1c_1} \vee \neg x_{2c_2} \vee \neg x_{3c_1} \vee \neg x_{4c_2}$.

Já a função `strategyFull` adiciona para cara atribuição uma cláusula unitária indicando se foi correta ou falsa.

O executável `mastermind.jar` recebe 3 parâmetros, nesta ordem: número de espaços, número de cores e número de tentativas. Assim, a seguinte chamada: `java -jar mastermind.jar 4 6 10` indica uma rodada do Mastermind com 4 espaços, 6 cores e 10 tentativas.

6 Experimentos

Além de implementar parte da estratégia do *codebreaker* na função `_strategy_new` ou `strategyNew`, vocês devem realizar o seguinte experimento: para cada $n \in \{4, 16, 64\}$ e cada $c \in \{6, 36, 108\}$, execute 10 vezes o Mastermind com $2.c$ tentativas. Anote em uma tabela o número de vezes em que o CodeBreaker ganhou, em outra tabela indique o número médio de tentativas que o CodeBreaker precisou para ganhar.

Repita o mesmo experimento substituindo sua estratégia por cada uma das estratégias iniciais oferecidas (*simple* e *full*). Você nota alguma diferença?

Escreva um breve relatório contendo: a descrição da nova estratégia implementada, diga quantas cláusulas sua estratégia produz e qual o tamanho delas em termos de n e c, as tabelas com os resultados dos experimentos para cada estratégia e uma conclusão comparando as estratégias.

7 Entrega

Entregue o código com sua versão modificada do exercício-programa e o relatório em PDF em um arquivo zip. O nome do arquivo zip deve seguir a seguinte regra: se seu nome for Fulano Ciclano Beltrano, o nome do arquivo deve ser: `ep1_fulano_ciclano_beltrano.zip`