Laboratórios de Informática II

Illuminatus — Etapa 3

Versão 1.1

2011/2012

Comandos a implememtentar

`rsv` Comando sem argumentos que deve resolver o tabuleiro no estado em que ele está;

`gera <ncols> <nlins>` Comando com dois argumentos, o nº de colunas e o nº de linhas que deve gerar um tabuleiro desse tamanho.

Só para que não restem dúvidas, ao executar o comando:

```
gera 38 20
```

O programa deverá gerar um tabuleiro com 38 colunas e 20 linhas com solução única.

Resolução do Puzzle

Tal como o nome indica, o objectivo é ser capaz de resolver um puzzle. Na prática isto implica tentar resolver o puzzle a partir da posição actual (assumindo que todas as marcas e lâmpadas colocadas estão correctas). Isto pode implicar que o puzzle não tenha solução. No caso de isso acontecer, deve-se devolver o erro correcto para indicar isso.

Sugere-se que a resolução aproveite ao máximo o que já foi feito no projecto nomeadamente a parte de anular comandos. Uma estrutura para o resolve deveria ser a seguinte:

1. Escolher uma casa promissora que ainda esteja livre;
2. Tentar colocar um dos dois valores possíveis nessa casa (e.g. lâmpada);
3. Aplicar as estratégias enquanto isso modificar o estado do puzzle;
4. Resolver recursivamente o puzzle;
5. Se a resolução do resto do puzzle foi bem sucedida sair da função reportanto sucesso;
6. Se foi descoberto um erro anular a operação anterior e experimentar o outro valor (e.g., marca) e voltar ao passo 2;
7. Se experimentámos ambos os valores e tivemos insucesso em ambos os casos então anular a operação anterior e sair da função reportanto insucesso.

Repare que este programa é simples de implementar porque nos passos 6 e 7 usamos a parte de anular comandos. A eficiência da resolução depende de muitas coisas mas sobretudo de uma escolha adequada de uma casa promissora no primeiro passo. Aqui deverá implementar a estratégia que preferir mas se quiser uma sugestão pense primeiro nas casas que determinem o maior número possível de casas do puzzle (ao invocar as estratégias)

Optimização de Código

Esta tarefa prende-se com utilizar as ferramentas `gcov` e `gprof` para analisar o código e descobrir as funções que estão a gastar mais tempo de CPU. Como seria de esperar, qualquer poupança nessas funções deverá ter um grande impacto (sobretudo aquelas que executam muitas vezes).

Esta tarefa será avaliada usando o vosso programa para resolver puzzles. Se o vosso programa resolver todos os puzzles que serão utilizados na avaliação então serão medidos os tempos de execução (sempre nas mesmas condições e na mesma máquina). O ponto dessa tarefa só será dado aos grupos que estejam num dado percentil dos melhores tempos.

Para usar essas ferramentas deverá compilar o código com as opções `-fprofile-arcs -ftest-coverage -pg` (mas lembre-se de as tirar quando for entregar). O código fica bastante mais lento já que está a analisar todas as linhas do código e a guardar essa informação em ficheiros.

O `gprof` mede os tempos (e.g. percentagem de tempo de CPU, nº de segundos) para cada função. O comando a invocar é `grof <nome do executável>`. Por exemplo, assumindo que o executável se chama `illuminatus`.

```
gprof illuminatus
```

O `gcov` conta o número de vezes que uma linha é executada. Ao contrário do anterior este comando deve ser executado para cada ficheiro fonte. Por exemplo, assumindo que existe um ficheiro chamado `interpretador.c` o comando a executar para analisar o nº de vezes que cada linha é executada seria:

```
gcov interpretador.c
```

Geração de Puzzles

A geração de puzzles deverá gerar puzzles:

- Com uma única solução;
- Que sejam interessantes para um ser humano.

O ser interessante para um humano implica que não seja necessário andar com a borracha se se estivesse a resolver num papel. Isto é, o nº de tentativas deve ser extremamente limitado para que um humano não sinta que anda a tactear sem ter a mínima noção de como se resolve o puzzle.

Análise do Código Gerado pelo Compilador

1. Escreva uma função (se ainda não a possui) que receba como parâmetro a estrutura de dados do tabuleiro e que conte quantas lâmpadas existem no tabuleiro (esta função deverá devolver um inteiro e vamos imaginar que se chama `contar_lampadas`);
2. Compile o código e seguidamente use o `gdb` (i.e., `gdb illuminatus`);
3. Vá buscar o código gerado pelo compilador para a função (i.e. `disassemble contar_lampadas`);
4. Crie a tabela de alocação de registos;
5. Identifique as instruções de assembly que implementar as estruturas de controlo (i.e., os ciclos);
6. Entregue um ficheiro com o resultado chamado `analise.pdf` juntamente com o resto do trabalho colocando este ficheiro na raiz (ao mesmo nível do ficheiro —verb—identificacao— e das pastas `code` e `doc`.