Especificação do Projeto Sat Solver 2019.1 - UFCA

Paola Accioly
Introdução à Ciência da Computação

6 de Junho de 2019

1 Introdução

O objetivo deste trabalho é praticar a escrita de funções e programas em Python, fazendo com que os alunos de primeiro período do curso de Ciência da Computação ganhem alguma familiaridade com uma linguagem bastante útil na prática e que, apesar da semelhança sintática, tem diferenças consideráveis com relação à linguagem C. Outro objetivo do projeto é apresentar aos alunos o problema SAT e a noção de intratabilidade. SAT é um dos problemas mais importantes da computação e a experiência de implementar uma solução, ainda que ineficiente, para tal problema, é bastante enriquecedora.

Para ter sucesso nesse projeto, a primeira tarefa é ler este documento com bastante calma e atenção para se certificar de que entendeu completamente o que é pedido, de forma a não perder tempo com idas e vindas ao documento e ter certeza de que de fato está entregando o que está especificado. Leia atentamente o texto, destacando partes importantes, anotando as dúvidas para saná-las o quanto antes. Além disso, para entender este projeto, será necessário ler documentação adicional indicada a seguir. A leitura de tal documentação é *fundamental*.

2 Visão Geral do Projeto

Neste projeto, você deverá construir um programa capaz de verificar se uma fórmula escrita em lógica proposicional é satisfatível. Uma fórmula lógica é satisfatível se é possível encontrar uma atribuição de valores V e F para suas variáveis de modo a tornar essa fórmula lógica verdadeira. A seguir são apresentadas duas fórmulas, uma satisfatível e outra não-satisfatível:

$$(A \vee B \vee \neg C) \wedge (\neg A \vee C) \wedge (B \vee \neg A \vee \neg C) \tag{1}$$

$$(\neg A \vee \neg C) \wedge (A \vee \neg B) \wedge (B \vee C) \wedge (\neg B \vee C) \wedge (B \vee \neg C) \tag{2}$$

A primeira é satisfatível se a A, B e C forem atribuídos os valores F, V e F, respectivamente. Há outras atribuições que também tornam a fórmula verdadeira. Já para a segunda, não há atribuição de valores que torne a fórmula verdadeira (verifique isso!). Este problema é conhecido como SAT e é um problema difícil na prática. Programas cujo objetivo é resolver instâncias de SAT são conhecidos como SAT solvers ou "solucionadores de satisfatibilidade". SAT é o que chamamos de um problema NP-completo. Para saber mais sobre problemas

NP-completos, você pode ler os dois textos referenciados abaixo. O primeiro é um email bastante didático escrito pelo professor Jorge Stolfi do IC-UNICAMP, enquanto o segundo é um tutorial sobre problemas NP-completos escrito pelo professor Paulo Feofiloff do IME-USP.

- E-mail do professor Jorge Stolfi;
- Artigo do professor Paulo Feofiloff.

Quem tiver mais dúvidas sobre problemas NP-Completos podem consultar o professor Thiago Marcilon aqui da UFCA (thiago.marcilon@ufca.edu.br).

A fórmula a ser verificada será lida a partir do teclado do usuário que deverá digitar as linhas codificadas de maneira pré-definida. As fórmulas contém apenas variáveis lógicas e os operadores AND ($\wedge$), OR ($\vee$) e NOT ($\neg$). Além disso, as fórmulas estão na forma normal conjuntiva (Conjunctive Normal Form – CNF[1]). Isso significa que estão organizadas em cláusulas onde variáveis lógicas (negadas ou não) são conectadas por operadores OR. As cláusulas, por sua vez, são conectadas apenas por operadores AND. Não há limite para a quantidade de variáveis em cada cláusula nem para a quantidade de cláusulas na fórmula. As duas fórmulas apresentadas anteriormente estão na CNF. A fórmula abaixo também está:

$$(\neg A \vee \neg C \vee D \vee \neg E) \wedge (A \vee \neg B \vee F) \wedge (B \vee C) \wedge (\neg D) \tag{3}$$

O projeto deverá ser desenvolvido na linguagem Python. Não é permitido implementar o projeto em nenhuma outra linguagem. Para desenvolver o projeto, você poderá usar como base o arquivo sat.py, disponível no repositório

- `https://github.com/prga/sat`

Tal arquivo fornece a estrutura geral que seu SAT solver deve apresentar. As funções que se encontram nesse arquivo, conforme indicam os comentários, precisam ser implementadas. No repositório também estão disponíveis exemplos que podem ser usados para testar seu SAT solver.

3 Formato de Entrada

Abaixo é apresentado um exemplo de como é a entrada a ser processada pelo seu SAT solver. As linhas da entrada abaixo podem ser lidas uma de cada vez.

```
p cnf 5 5
1 2 3 4 5 0
-1 -2 -3 -4 -5 0
1 -2 3 -4 5 0
5 0
-1 2 0
```

A primeira linha a ser digitada pelo usuário é a linha de "problema". Sua sintaxe é definida abaixo.

[1] `https://en.wikipedia.org/wiki/Conjunctive_normal_form`

`p cnf #vars #clausulas`

onde `#vars` é o número de variáveis e `#clausulas` é o número de cláusulas. Para esse exemplo, a linha de problema é `p cnf 5 5`, o que significa que a fórmula que ele representa tem cinco cláusulas e cinco variáveis.

Como explicado na Seção 2, uma fórmula na CNF (Conjunctive Normal Form) consiste em cláusulas conectadas por operadores lógicos $\wedge$. Cada cláusula, por sua vez, consiste em sequências de variáveis, negadas ou não, conectadas por operadores lógicos $\vee$ (veja os exemplos na Seção 2). Devido a essa estrutura, o restante das linhas digitadas estão organizadas da seguinte maneira:

1. Variáveis são representadas por números consecutivos começando em 1. O exemplo anterior tem cinco variáveis: 1, 2, 3, 4 e 5. Números negativos representam variáveis que estão aparecendo negadas (o operador '-' representa o '$\neg$');

2. Cláusulas são sequências de variáveis, negadas ou não, finalizadas pelo número 0;

3. Não é necessário indicar explicitamente o '$\wedge$' ou '$\vee$'. Variáveis em uma mesma cláusula são conectadas implicitamente por '$\vee$' enquanto cláusulas diferentes são conectadas implicitamente por '$\wedge$'.

Para processar a entrada do usuário, implemente a função `readFormula`. Essa função deve ler as entradas do usuário e deve devolver um dicionário com dois elementos, `clauses` e `variables`. O primeiro deve guardar uma lista de cláusulas, onde cada cláusula é ela própria representada como uma lista (que pode ser de números ou de strings). Já o elemento `variables`, como o nome indica, deve guardar uma lista contendo uma quantidade de posições igual ao número de variáveis, onde a posição 0 corresponde à variável 1, a posição 1 à variável 2, a posição 2 à variável 3, etc. O valor guardado em cada posição pode ser uma atribuição inicial de valores a essas variáveis, por exemplo, 0 ou False. Para o exemplo acima, essa função pode devolver um dicionário como o seguinte:

```
{ clauses:  [[ '1', '2', '3', '4', '5' ],
[ '-1', '-2', '-3', '-4', '-5' ],
[ '1', '-2', '3', '-4', '5' ],
[ '5' ],
[ '-1', '2' ]],
variables:  [0, 0, 0, 0, 0]
}
```

Para realizar esta tarefa, sugere-se que sejam criadas três funções auxiliares: `readInputs` que processa os inputs do usuário e retorna uma lista com as linhas digitadas, `readClauses`, que extrai as cláusulas das entradas do usuário e produz uma lista como a armazenada no elemento `clauses` acima e `readVariables`, que processa as cláusulas e identifica todas as variáveis (ignorando se aparecem negadas ou não) e produz como resultado uma lista como a armazenada no elemento `variables` acima.

4 Atribuindo Valores

Este SAT solver deve resolver o problema de satisfatibilidade usando uma estratégia de força bruta. Isso significa que deve testar todas as possíveis atribuições de valores V e F às variáveis da fórmula, verificando para cada uma delas se o resultado da avaliação da fórmula inteira é verdadeiro. Essa estratégia não é nada eficiente, mas tem a vantagem de ser simples de implementar. Para esta seção, uma função a mais precisará ser implementada. Ela é responsável por gerar uma atribuição de valores booleanos para as variáveis da fórmula. Uma atribuição é uma lista onde cada posição corresponde a um valor (V ou F) atribuído a uma variável. A maneira como os valores V e F são representados é deixada a seu critério. No exemplo da seção anterior, usamos o número 0 para representar F, mas seria igualmente válido usar False.

Para realizar essa tarefa, implemente a função `nextAssignment`. Essa função recebe como entrada a lista contendo a atribuição atual de valores às variáveis da fórmula e devolve uma nova lista (ou a mesma lista, modificada) com uma nova atribuição de valores a essas variáveis. Se você perceber que fazer a atribuição de valores *in-place*[2] pode ser mais eficiente, você pode adotar essa estratégia. A função `nextAssignment` deve funcionar de modo que sempre produza atribuições diferentes de valores, para um determinado número de variáveis. Observe o exemplo a seguir, usando a linha de comando do Python para ilustrar e com os valores V e F representados por True e False, respectivamente:

```
> nextAssignment([False, False])
[False, True]
> nextAssignment([False, True])
[True, False]
> nextAssignment([True, False])
[True, True]
> nextAssignment([True, True])
[False, False]
```

Neste exemplo, todas as possíveis combinações de valores booleanos para essa lista representando uma fórmula com apenas duas variáveis foram geradas.

5 Verificando a Satisfatibilidade

Uma vez munidos das cláusulas, das variáveis e de uma atribuição de valores para estas últimas, podemos verificar se a fórmula é ou não satisfeita por essa atribuição de valores às suas variáveis, ou seja, se a avaliação da fórmula produz V ou não. Se for o caso, o programa termina, informa o usuário que a fórmula é satisfeita e qual foi a atribuição de valores identificada. Caso contrário, obtém a próxima atribuição usando `nextAssignment` e repete o procedimento até que a fórmula seja satisfeita ou todas as atribuições sejam verificadas. Nesta segunda situação, o programa informa o usuário que a fórmula não é satisfatível.

Para realizar essa tarefa, implemente a função `doSolve`. Essa função recebe como entrada a lista contendo a atribuição inicial de valores às variáveis da fórmula e outra lista contendo

[2]Ou seja, modificando diretamente o que já está na lista recebida como parâmetro, ao invés de criar uma nova lista e devolvê-la.

as cláusulas. A função deve funcionar em um loop onde, enquanto a fórmula não for satisfeita nem todas as atribuições tiveram sido testadas, ela verifica se a atribuição atual satisfaz ou não a fórmula e, se não satisfizer, pega a próxima atribuição de valores e tenta novamente. Seu resultado é um dicionário com dois elementos, (i) `isSat`, que guarda um valor booleano indicando se a fórmula foi ou não satisfeita, e (ii) `satisfyingAssignment`, que guarda `null` se a fórmula não foi satisfeita ou uma lista contendo a atribuição que a satisfaz, caso contrário.

6 Agrupando Tudo

Seu programa deve ser organizado como um módulo que oferece externamente uma função chamada `solve`. O esqueleto do projeto que está disponível no repositório já está organizado desta forma. É necessário, porém, implementar a função `solve` para que utilize as outras funções que você construiu.

7 Avaliação e Logística

O projeto representará 50% da nota de cada aluno na disciplina. Não entregar o projeto implica imediatamente em nota 0 (ZERO) para os membros da equipe que não entregou. A avaliação se dará através de apresentação do projeto desenvolvido para a professora. Ausência no momento da apresentação do projeto implica em nota 0 (ZERO) para todos os membros faltosos da equipe (com raras exceções). Algumas considerações adicionais, relativas à logística do projeto:

- O trabalho poderá ser feito em grupos de no máximo dois componentes;
- O código completo do programa deverá estar disponível em um repositório no GitHub (http://www.github.com). Deve ser enviado um `email` para o professor com assunto PROJETO SAT SOLVER até às 10h00 da manhã do dia 27/06/19 informando o endereço do repositório. Programas não entregues no horário serão penalizados com a perda de pontos, à taxa de 2,0 pontos perdidos para cada 24h de atraso;
- A apresentação dos projetos será durante as aulas dos dias 27/06/19 e 02/07/19. No começo da aula do dia 27/06/2019 será realizado um sorteio para definir a ordem das apresentações. Todos os membros devem saber todos os detalhes da implementação. A professora escolherá na hora a quem dirigirá cada pergunta durante a apresentação.