3-SAT

Bruno de Morais Bueno
Prof. a Mariana Kolberg
INF05515 - Complexidade de Algoritmos

FNC (Forma Normal Conjuntiva)

FNC - Expressão booleana com operações de conjunção, disjunção e negação:

- É constituída por uma conjunção de uma ou mais cláusulas.
- Cada cláusula é constituída por disjunções de um ou mais literais;
- Um literal é uma ocorrência da variável, podendo ser a própria variável ou seu complemento.

Problema - SAT

Problema da Satisfabilidade Booleana (Boolean Satisfiability Problem - SAT).

Problema de decisão e primeiro provado NP-completo.

Dada uma expressão na FNC, ele questiona se existe uma combinação de atribuições das variáveis de entrada que torne a expressão satisfazível.

***As cláusulas da expressão não possuem um número determinado de literais e literais de mesma variável.*

Problema - 3-SAT

Assemelha-se ao Problema SAT, porém, cada cláusula da FNC possui **somente** 3 literais.

Dada uma FNC *composta por somente 3 literais em cada cláusula*, ele questiona se existe uma combinação de atribuições das variáveis de entrada que torne a expressão satisfazível.

3-SAT pertence a NP ?

Para realizar a prova de que 3-SAT pertence a NP, deve-se apresentar um algoritmo de verificação que:

- dada uma expressão constituída por cláusulas na FNC com 3 literais cada (instância);
- dado um conjunto de atribuições para cada variável da expressão(certificado);
- em tempo polinomial, retorna se a avaliação da expressão é satisfazível ou não;

Algoritmo de verificação

instancia = $(x_0 \vee x_1 \vee x_2) \wedge (!x_0 \vee !x_1 \vee x_3) \wedge (!x_0 \vee !x_3 \vee !x_2)$

certificado = {true, false, false, false}

$$(\text{true} \vee \text{false} \vee \text{false}) \wedge (\text{false} \vee \text{true} \vee \text{false}) \wedge (\text{false} \vee \text{true} \vee \text{true})$$

$$(\text{true} \wedge \text{true} \wedge \text{true})$$

$$\text{true}$$

Algoritmo de verificação

```
cl1 = ["x0","x1","x2"]
cl2 = ["!x0","!x1","x3"]
cl3 = ["!x0","!x3","!x2"]

instancia = [cl1,cl2,cl3]
certificado = [True,False,False,False]
```

```
aux2 = True

for clausula in instancia:
    aux = False
    for literal in clausula:
        if(literal[0] == "!"):
            aux = aux or not(certificado[int(literal[2])])
        else:
            aux = aux or certificado[int(literal[1])]
    aux2 = aux2 and aux

print(aux2)
```

N = n° total de cláusulas da expressão

Custo constante
O(3) ou O(1)

Custo
O(3N) ou O(N)

3-SAT pertence a NP-Difícil ?

Para provar que um problema pertence à classe dos problemas NP-Difíceis parte-se de qualquer instância de um problema já conhecido e provado NP-Completo e obtém-se, em tempo polinomial, uma instância do problema que se deseja provar, de forma que as respostas sejam equivalentes(Redução).

Neste caso, para provar que 3-SAT pertence a NP-Difícil será realizada a redução do problema SAT para 3-SAT, transformando todas as cláusulas de SAT para que possuam exatamente 3 literais em cada e ambos os problemas tenham igual resposta.

Redução

Para cada cláusula da expressão SAT, cria-se a devida equivalência na nova expressão 3SAT da seguinte forma:

Se cláusula tem apenas 1 literal:

Cria-se duas novas variáveis e adicionam-se quatro cláusulas para a nova expressão com:

- literal (SAT);
- combinações variando as possíveis atribuições das novas variáveis.

***Quando o único literal(SAT) for verdadeiro todas as novas cláusulas serão verdadeiras, quando falso ao menos 1 será falsa.*

$$(x_3)$$

$$(x_3 \vee z_0 \vee z_1) \wedge (x_3 \vee z_0 \vee \neg z_1) \wedge (x_3 \vee \neg z_0 \vee z_1) \wedge (x_3 \vee \neg z_0 \vee \neg z_1)$$

Redução

Se cláusula tem apenas 2 literais:

Cria-se uma nova variável e adicionam-se duas cláusulas para a nova expressão com:

- 2 literais da expressão SAT;
- e combinação com as 2 possíveis atribuições para a nova variável.

**Quando ao menos 1 literal dos literais oriundos da expressão SAT for verdadeiro todas as novas cláusulas serão verdadeiras, quando ambos forem falsos ao menos 1 será falsa.*

$$(x_1 \vee !x_3)$$

$$(x_1 \vee !x_3 \vee z_0) \wedge (x_1 \vee !x_3 \vee !z_0)$$

Redução

Se cláusula tem mais de 3 literais:

$$(!x_0 \text{ V } !x_1 \text{ V } x_2 \text{ V } !x_3 \text{ V } x_4 \text{ V } x_5)$$

Criam-se |cláusula|-3 novas variáveis e adicionam-se |cláusula|-2 cláusulas para a nova expressão de forma que:

1° nova cláusula é composta por:

- 2 primeiros literais da cláusula SAT;
- 1 literal correspondente a uma variável nova.

$$(!x_0 \vee !x_1 \vee x_2 \vee !x_3 \vee x_4 \vee x_5)$$

$$(!x_0 \vee !x_1 \vee z_0) \wedge \ldots$$

2° a penúltima cláusula é composta por:

- último literal da cláusula anterior negado;
- 1 literal da expressão SAT variante (3° literal até antepenúltimo)
- 1 literal correspondente a uma variável nova.

$$(!x_0 \vee !x_1 \vee z_0) \wedge (!z_0 \vee !x_1 \vee z_1) \wedge (!z_1 \vee x_2 \vee z_2) \wedge \ldots$$

Última cláusula é composta por:

- último literal da cláusula anterior negado;
- 2 últimos literais da cláusula SAT;

$$(!x_0 \vee !x_1 \vee x_2 \vee !x_3 \vee x_4 \vee x_5)$$

$$(!x_0 \vee !x_1 \vee z_0) \wedge (!z_0 \vee !x_1 \vee z_1) \wedge (!z_1 \vee x_2 \vee z_2) \wedge (!z_2 \vee x_4 \vee x_5)$$

Redução

Para quê? Para manter a cláusula satisfazível de forma que:

- Se $literal_1$ ou $literal_2$ da cláusula SAT são verdadeiros, atribui-se valor falso a todas as novas variáveis;
- Se $literal_{n-1}$ ou $literal_n$ da cláusula são verdadeiros, atribui-se valor verdadeiro a todas as novas variáveis
- Se $literal_i$ é verdadeiro ($2 < i < n-1$), atribui-se valor verdadeiro as variaveis anteriores a $literal_i$ e valor falso as variaveis posteriores a ele
- Se todos literais são falsos haverão várias cláusulas falsas insatisfazendo toda expressão.

Algoritmo de redução

```
cl1 = ["x0"]
cl2 = ["!x0","!x1","x2","!x3","x4"]
cl3 = ["!x0","!x3"]
cl4 = ["x1","x3","!x4"]

expressao = [cl1,cl2,cl3,cl4]
```

```
for clausula in expressao:
    if len(clausula) == 1:
        aux = [[clausula[0],("z" + str(i)),("z" + str(i+1))],
        [clausula[0],("z" + str(i)),("!z" + str(i+1))],
        [clausula[0],("!z" + str(i)),("z" + str(i+1))],
        [clausula[0],("!z" + str(i)),("!z" + str(i+1))]]
        novaExpressao = novaExpressao + aux
        i = i + 2

    elif len(clausula) == 2:
        aux = [[clausula[0],clausula[1],("z" + str(i+1))],
        [clausula[0],clausula[1],("!z" + str(i+1))]]
        novaExpressao = novaExpressao + aux
        i = i + 1
    elif len(clausula) == 3:
        novaExpressao.append(clausula)
```

Custo constante
O(1)

Custo constante
O(1)

Custo constante
O(1)

Algoritmo de redução

```
else:
    aux = []
    ultimo = len(clausula) - 1
    auxPrimeira = [clausula[0],clausula[1],("z" + str(i))]
    for x in range(2,ultimo-1):
        aux = aux + [("!z" + str(i)),clausula[x],("z" + str(i+1))]
        i = i + 2
    i = i - 1
    auxUltima = [("!z" + str(i)),clausula[ultimo-1],clausula[ultimo]]

    novaExpressao = [auxPrimeira] + novaExpressao + [aux] + [auxUltima]
```

}

Custo
O(L-4) ou O(L)

L = n° de literais da cláusula

Algoritmo de redução

```
for clausula in expressao:
    if len(clausula) == 1:
        aux = [[clausula[0],("z" + str(i)),("z" + str(i+1))],
        [clausula[0],("z" + str(i)),("!z" + str(i+1))],
        [clausula[0],("!z" + str(i)),("z" + str(i+1))],
        [clausula[0],("!z" + str(i)),("!z" + str(i+1))]]
        novaExpressao = novaExpressao + aux
        i = i + 2

    elif len(clausula) == 2:
        aux = [[clausula[0],clausula[1],("z" + str(i+1))],
        [clausula[0],clausula[1],("!z" + str(i+1))]]
        novaExpressao = novaExpressao + aux
        i = i + 1
    elif len(clausula) == 3:
        novaExpressao.append(clausula)
    else:
        aux = []
        ultimo = len(clausula) - 1
        auxPrimeira = [clausula[0],clausula[1],("z" + str(i))]
        for x in range(2,ultimo-1):
            aux = aux + [("!z" + str(i)),clausula[x],("z" + str(i+1))]
            i = i + 2
        i = i - 1
        auxUltima = [("!z" + str(i)),clausula[ultimo-1],clausula[ultimo]]

        novaExpressao = [auxPrimeira] + novaExpressao + [aux] + [auxUltima]
```

Custo
O(L-4)

L' = n° de literais da maior cláusula
N = n° total de cláusulas da expressão

Custo total do pior caso:
O(N * (L' - 4)) ou O(N*L')

Conclusão

Problema 3SAT é NP-Completo !