Teoria da computação Q2.2018 - Lista 3

João Carlos Pandolfi Santana

August 2018

1 Problema 1

Assumindo que o problema de satisfatibilidade (SAT) é NP-completo, prove que os seguintes problemas também são NP-completo: 3SAT, Clique, Independent set, Vertex Cover, Subset Sum, Knapsack, Partition, Longest Path, Rudrata Path, Traveling Salesman e Integer Linear Programming.

É sabido que:

- Uma variável booleana *barrada* ou não, é um **literal**.
- Uma **cláusula** é composta por um ou mais literais conectados por $\vee$ (*ou*s).
- Uma expressão booleana está na Forma Normal Conjuntiva (**FNC**) se, e somente se, for composta de uma ou mais cláusulas conectadas por $\wedge$ (*e*s).
- Uma expressão booleana é uma *fórmula-FNC* se estiver na Forma Normal Conjuntiva
- SAT é uma formula-FNC satisfazível
- SAT é NP-Completo
- A prova de um problema K NP-Completo é dividido em duas partes. **(i)** - Provar que K é NP. **(ii)** - Provar que todo problema NP pode ser reduzido para K em tempo polinomial. (*Teorema de Cook-Levin para SAT*)
- Outra forma de provar um problema K é NP-Completo segue em duas partes. **(i)** - Provar que K é NP. **(ii)** - Reduzir algum problema NP-Completo para K em tempo polinomial

1.1 3SAT

Prova de NP-Completude para 3SAT

1. Mostrar que **3SAT** está em NP
2. Ao tomar como premissa SAT como NP-Completo, reduzir SAT para $3SAT$ em tempo polinomial prova que este é NP-Completo
3. Para fazer a redução de SAT, converte-se a fórmula-FNC F para fórmula-3FNC F', de forma que: sendo F satisfazível $\Leftrightarrow$ F' é satisfazível.

- Assumimos que C_1, C_2, C_3..., C_n são cláusulas de F onde F está descrita na fórmula 3NFC (cada cláusula C_i deve haver 3 ou menos literais), assumimos que F' é F.
- Em cada cláusula (C_i) que possuir 1 ou dois Literais (L_i), descrita da seguinte forma (L_1). Replicamos o L_1 conectados por $\vee$ até a cláusula C_i conter 3 Literais. Ex.: Dado F=($L_1 \vee L_2$), aplicamos o "algoritmo"e obtemos F'=($L_1 \vee L_2 \vee L_1$).
- Assim se F' é satisfazível, o valor de L_1 deve ser igual a 1 (*true*)
- Para C_i onde a quantidade de Literais é igual a 3, nenhuma mudança é aplicada.
- Para C_i=($L_1 \vee L_2$,$L_3 ... \vee L_n$) onde $n > 3$, reescrevemos da seguinte forma: $(L_1 \vee L_2 \vee Y_1) \wedge (\neg Y_1 \vee L_3 \vee Y_2) ... \wedge (\neg Y_k \vee L_{n-1} \vee L_n)$.
- Desta forma, se F' é satisfazível, o valor de C_i deve ser igual a 1 (*true*)

1.2 Clique

Prova de NP-Completude para Clique

1. Como foi provado, *3SAT* é NP-Completo, portanto, vamos reduzir o *3SAT* para *Clique*.
2. Se esta conversão for possível em tempo polinomial, o problema *Clique* é NP-Completo.
3. Tomando F como uma fórmula-3CNF, sendo C_1, C_2,..,C_n **cláusulas** de F e L_1,L_2,...,L_m **literais** de C_k.
4. Construímos o grafo G, tal que: F é satisfazível $\Leftrightarrow$ G é satisfazível.
   - Prova por contraposição
   - Para cada literal $L_{a,b}$ é criado um **vértice** representativo em G.
   - G contém todos as arestas, exceto se:
     (a) Há dois vértices representando a mesma cláusula
     (b) Há dois vertices conectados sendo estes a respectiva negação do literal. Ex.: L_c conectado com $\neg L_c$
5. Temos que: G é um *k-clique* $\Leftrightarrow$ F é satisfazível
   - Se G for um *K-clique* tem que acontecer:
     (a) O *K-clique* deve ter um uma aresta para cada cláusula
     (b) Nenhum vértice será a negação de outro no *clique*
   - Portanto, quando for setado o correspondente literal para 1 (*true*), F será satisfazível.
6. Se F for satisfazível, pelo menos um literal em cada cláusula é definido como 1 (*true*) no espaço satisfazível.
7. Então os vértices satisfazíveis correspondem a um *clique*. Então G tem um *K-clique*.
8. Logo se G puder ser construído a partir de F em tempo polinomial do *3SAT* para o *Clique*, temos que o problema *Clique* é **NP-Completo**

1.3 Independent set

Prova de NP-Completude para Independent Set

1.4 Vertex Cover

Prova de NP-Completude para vertex Cover

1.5 Subset Sum

Prova de NP-Completude para *Subset Sum*.

1. Como foi provado, *3SAT* é NP-Completo, portanto, vamos reduzir o *Subset Sum* para *3SAT* em tempo polinomial.

2. Supondo que A está descrito na fórmula-3NFC e que l_1,l_2,l_3...,l_n são literais e que C_1,C_2,C_3,...,C_m são as cláusulas. Transforma-se A em um um novo conjunto P de $2n + 2m$ elementos numéricos, de forma que cada m contenha $n + m$ dígitos.

3. Para cada literal l_i, são definidos dois números y_i e z_i, onde o iésimo digito será 1.

4. *(i)* se l_i estiver em C_k, define-se o (n+k) ésimo digito de y_i, até 1, *(ii)* se $\neg x_i$, estiver em C_m, define-se o (m+k) ésimo dígito de z_i até 1. Os outros dígitos de y_i e z_i serão setados a 0.

5. A deve conter um par de números iguais, g_k e h_k para cada cláusula em C_k, onde $(n + k)$ ésimo dígito é setado para 1.

6. Seja t = 111...13333...3 [j '1' seguidos por k '3'] número alvo. É preciso mostrar que F é satisfazível $\Leftrightarrow$ algum subconjunto de S adiciona t. Supondo que F seja satisfazível, então seleciona-se y_i se x_i for *verdadeiro*. Caso contrário, selecionamos z_i. Se adicionarmos os números definidos até então então, *(i)* Os dígitos j à esquerda estão corretamente posicionados e *(ii)* cada um dos k dígitos restantes está entre 1 e 3. Próximo passo é selecionar um g_k e/ou h_k adequado para preencher as diferenças, chegando portanto em t.

7. Ao supormos que um subconjunto S some t. Então, podemos dizer que exatamente um dos y_i ou z_i está presente neste subconjunto. Definindo x_i **verdadeiro** quando y_i está no subconjunto, e **falso** quanto z_i está no subconjunto, então podemos dizer que F foi satisfeito, de forma a considerarmos os números no subconjunto cujo $(n + k)$ iésimo bit é 1.

8. Podem haver 3 desses números, portanto, algum deles deve respeitar a regra do y_i ou z_i. Caso seja y_i, significa que C_k contém x_i e deve ser assinalado como **verdadeiro**. Caso seja z_i significa que C_k contém $\neg x_i$ e deve ser assinalado como **falso**. Em ambos os casos C_k foi satisfeito.

9. Desta forma, com todas as Cláusulas e F satisfeitos, mostramos que $(F) \in 3SAT \Leftrightarrow (St, t) \in$ *Subset-Sum*. Assim, Subset-Sum é NP-Completo.

1.6 Partition

Prova de NP-Competude para Partition

1. Como provamos que *Subset-sum* é NP-Completo, basta reduzirmos *Partition* para *Subset-sum*.
2. Para determinar se S,k está em Subset-sum, definimos que X é a soma dos valores em S. Construímos S' adicionando dois números $2X - k$ e $X + k$ a S.
3. Assim $(S, k) \in$ *Subset-sum* $\Leftrightarrow (S') \in$ *Partition*. Desta forma, Partition é NP-Completo.

1.7 Knapsack

Prova de NP-Competude para Knapsack

1. Como provamos que Partition é NP-Completo, basta reduzir Knapsack em tempo polinomial para NP-Completo.
2. Estabelecemos $S = \{s_1, s_2, ..., s_j\}$ como um conjunto de inteiros positivos.

1.8 Longest Path

1.9 Rudrata Path

1.10 Traveling Salesman

1.11 Integer Linear Programming

2 Problema 2

O problema de encontrar todos os fatores primos de um dado número inteiro é NPcompleto? Ele pertence à classe NP?

- Não, atualmente ainda não se sabe se o problema da Fatoração está em *NP-Completo*, e até agora não conseguiram provar que está em *P*.
- Sim, ele pertence a classe *NP*

3 Problema 3

Arquivo com a implementação dos problemas em anexo.
np-complete.py