UNIVERSIDADE FEDERAL DE OURO PRETO
Rafael Francisco de Oliveira - 2021.10171
PCC104 - Projeto e Análise de Algoritmos
Github: raffoliveira
Lista 5

---

1. O que significa dizer que um algoritmo resolve um problema em tempo polinomial?

   Significa que a eficiência de um algoritmo no seu pior caso pertence a classe $O(p(n))$ onde $p(n)$ é uma função polinomial de acordo com o tamanho da entrada n, ou seja, a complexidade de tempo do algoritmo progride lentamente de acordo com uma enorme tamanho de entrada n. Por exemplo, $O(c), O(\log n), O(N^c), O(\sqrt{N}), O(n^2)$ são funções do tipo $O(n^k)$ limitada superior por n^k para qualquer constante k.

2. Que tipo de problemas considera-se tratável?

   Problemas que podem ser decididos e resolvidos em tempo polinomial, ou seja, para um problema de tamanho n, o tempo ou o número de etapas necessárias para encontrar a solução é uma função polinomial de n. O limite superior é uma polinomial. Exemplos: busca em lista não ordenada e ordenada, ordenação de conjuntos, número primo,

3. Que tipo de problema considera-se intratável?

   Problemas que não podem ser decididos e resolvidos em tempo polinomial, ou seja, requerem tempos que são funções exponenciais do tamanho do problema n. O limite inferior é uma exponencial. Na instância de complexidade computacional, são problemas que não existe algoritmos eficientes para resolvê-los. Exemplos: torre de Hanoi, geração de todas permutações de n números, fatoração de número primos.

4. Em ciência da computação, o que é o conjunto ou classe de problemas P?

   Classe P ou polinomial é uma classe de problemas de decisão, que são problemas que englobam respostas sim/não, e que são resolvidos em tempo polinomial por algoritmos determinísticos (dada uma entrada sem produzirá a mesma saída). Temos como exemplos, máximo divisor comum, ordenação de conjuntos, busca de chaves ou padrões e conectividade ou ciclos em grafos.

5. Como podemos provar que um problema pertence à classe P?

   Formalmente:

$$L \in \mathbf{P} \left| \begin{array}{l} \text{iff } \exists M \\ M \in O(n^k) \text{ para algum } k \\ \forall x \in L, M(x) = 1 \\ \forall x \notin L, M(x) = 0 \end{array} \right.$$

   Essencialmente, um problema será **P** se, e somente se, se pudermos resolvê-lo em um computador normal em tempo polinomial. Logo, é necessário a construção de um algoritmo e a prova que ele executa em tempo $O(n^k)$, com k constante. Para a prova, pode-se utilizar simulação, transformação ou redução de linguagens.

6. De forma geral, o que é um algoritmo determinístico?

   Algoritmo no qual, dada uma entrada, irá produzir a mesma saída em todas as execuções seguindo as mesmas etapas de processamento. Formalmente, um algoritmo determinístico computa uma função matemática tendo um único valor como entrada e um único valor como saída. A maioria dos algoritmos estudados são deste tipo devido sua eficiência.

7. De forma geral, o que é um algoritmo não determinístico?

   Neste caso, o algoritmo, dada uma entrada, irá produzir saídas diferentes a cada execução. Na verdade, algoritmos não determinísticos não podem resolver o problema em tempo polinomial e não podem determinar qual é a próxima etapa. Os algoritmos não determinísticos podem mostrar comportamentos diferentes para a mesma entrada em execuções diferentes e há um certo grau de aleatoriedade nisso.

8. Em ciência da computação, o que é o conjunto ou classe de problemas NP?

   Classe NP ou *Non-Deterministic Polynomial time* é uma classe de problemas de decisão que podem ter seu certificado verificado em tempo polinomial por uma máquina de Turing determinística, ou seja, os problemas podem ser resolvidos por um algoritmo não-determinístico polinomial. Ainda, pode-se dizer que é uma classe com verificadores eficientes, ou seja, há um algoritmo de tempo polinomial capaz de verificar se dada uma solução é correta ou não.

9. O que é um algoritmo polinomial não determinístico?

Um algoritmo não-determinístico consiste em duas etapas: a primeira consiste em ter um palpite sobre a possível solução do problema proposto, geralmente gerada por maneira não-determinística. A segunda etapa consiste em construir um algoritmo determinístico para verificar se o palpite é a solução do problema. Logo, um algoritmo não-determinístico é dito polinomial não determinístico se a eficiência de tempo da etapa de verificação for polinomial.

10. Explique por quê $P \subseteq NP$?

    A classe P (classe que engloba todos os problemas solucionáveis, deterministicamente, em tempo polinomial) está contida ($\subseteq$) na classe NP (classe que engloba os problemas onde a solução pode ser verificada em tempo polinomial) porque se um problema pode ser resolvido em tempo polinomial, então uma solução também é verificável em tempo polinomial simplesmente resolvendo o problema.

11. Por quê saber se $P = NP$ é interessante?

    A expressão diz se existe algoritmos de tempo polinomial que resolvem problemas NP-completos, e por corolário, todos os problemas NP. Esse é um dos maiores problemas ainda não resolvidos pela computação. Caso a resposta fosse positiva, seria um grande salto para a computação pois muitos problemas difíceis e aplicáveis ao cotidiano seriam solucionados em tempo polinomial, como por exemplo, o problema do caixeiro viajante aplicado em problemas de logística.

12. Como provamos que um problema é NP-Completo?

    Primeiramente devemos mostrar que: (1) pertence a classe NP e (2) é um NP-difícil. Para provar (1) pode utilizar a estratégia de verificação de certificado, onde o certificado é uma resposta do problema. O verificador é um algoritmo de tempo polinomial que irá verificar se a resposta do problema é "Sim" ou "Não". Só é possível continuar com a prova se (1) for provado verdadeiro. Para provar (2) utiliza-se a estratégia de redução polinomial, que consiste em utilizar outro problema NP-difícil e reduzi-lo ao problema atual. Vale ressaltar que todo NP-completo é NP-difícil, mas o contrário não. Assim, pode reduzir também um NP-completo.

    A redução de um problema X para um problema Y ($X \leqslant_p Y$) é uma conversão de entradas do problema X para entradas do problema Y. A conversão é um algoritmo de tempo polinomial com a complexidade dependente do tamanho da entrada. As entradas do problema de decisão são "Sim" e "Não". Se $X \leqslant_p Y$ então toda entrada "Sim" de X se converte para toda entrada "Sim" de Y. O mesmo acontece para entrada "Não". A redução é transitiva, então se A reduz

B com um algoritmo α e B reduz para C com um algoritmo β, então A reduz para C com um algoritmo $\alpha \circ \beta$.

13. Como provamos que um problema é NP-Completo quando já conhecemos algum problema NP-Completo?

    Dado que o problema é T, a ideia é pegar o problema NP-Completo conhecido e reduzi-lo a T. Se a redução de tempo polinomial for possível, pode-se provar que T é NP-Completo por transitividade de redução (Se um problema NP-Completo é redutível a T no polinômio tempo, então todos os problemas são redutíveis a T em tempo polinomial).

14. O que significaria resolver ao problema NP-Completo em $O(n^5)$?

    Isto indicaria que o problema NP-completo está sendo resolvido em tempo polinomial, ou seja, NP-completo está em P. Logo $P = NP$, um dos maiores problemas na computação estaria solucionado.

15. Um algoritmo que faz um número polinomial de chamadas a um procedimento que executa em tempo polinomial pode ter complexidade exponencial? Explique.

    Não. Por exemplo, se as chamadas tem complexidade n^2 e o procedimento tem complexidade n^5, então a complexidade total seria $n^2 * n^5 = n^7$. Logo, seria polinomial e não exponencial. Uma vez que polinomial tem como base a entrada do problema (n^k) e exponencial tem como expoente a entrada do problema (k^n), então não teria como transformar polinomial em exponencial.

16. Qual dos diagramas abaixo não contradiz o estado corrente do nosso conhecimento sobre as classes de problemas P, NP e NP-Completo.

    Letra E. As letras A, B e D implicam em N=NP, o que não é válido ainda. A letra C implica que todos os problemas NP são ou P ou NP, porém há problemas NP's que não é nenhum nem outro.

17. Mostre que o Problema do Conjunto independente (IS) é um problema NP-Completo utilizando a redução entre problemas, considerando o 3-SAT como problema base.

    Problema do Conjunto Independente (G, k): Dado um grafo G e um número natural k, existe um conjunto de k vértices independente (ou seja, sem arestas entre si) em G?

IS está em NP: a partir de um grafo G e um número natural k, se for dado um conjunto S de k vértices, podemos verificar em tempo polinomial no tamanho de G se S é um conjunto independente, ou seja, um conjunto independente de k vértices de G é um certificado polinomial para o IS. Se G for dado como matriz de adjacência, então pode ser feito em $O(n^2)$. Portanto, IS está em NP.

Reduzir 3-SAT em IS (3-SAT $\leqslant_p$ IS): dada uma instância para o 3-SAT, ou seja, uma coleção de cláusulas C em que todas elas têm no máximo 3 literais (um é verdadeiro e os outros dois são falsos), é preciso transformá-la em uma instância de IS, ou seja, um grafo G e um número natural k. Para isso, deve assegurar que:

- Um algoritmo para a transformação de C em G e k deve ser polinomial no tamanho de C.
- 3-SAT(C) deve ter resposta *sim* se, e somente se, IS(G, k) tiver resposta *sim*.

Primeiro, definimos k como o número de cláusulas de C. A cada literal de C, associamos um vértice em G.

As arestas de G são definidas da seguinte maneira: a cada cláusula (abc) de C, associamos as arestas $\{a, b\}, \{a, c\}$ e $\{b, c\}$. E sempre que uma variável x e sua negação $\neg x$ aparecem em C, criam-se uma aresta $\{x, \neg x\}$.

O número de vértices de G é igual ao número de literais em C. O número de arestas em G definidas por cada cláusula é, no máximo, o número de literais em C. O número de arestas em G definidas pelas variáveis e suas negações não ultrapassa o número de literais ao quadrado. Assim, a transformação de C em G e k é polinomial no tamanho de C.

Se 3-SAT(C) tem resposta *sim*, existe uma valoração que faz com que pelo menos um literal em cada cláusula de C seja verdadeiro. Para cada cláusula de C, considere apenas um literal verdadeiro nesta valoração. Seja S o conjunto formado pelos vértices em G associados a estes literais. Como S contém apenas um vértice associado a um literal de cada cláusula de C, $|S| = k$. Para cada par de vértices de S, não há arestas em G. Ou seja, S é um conjunto independente de G de tamanho k. Portanto, neste caso, IS(G, k) tem resposta sim.

IS é NP-completo. Para determinar o conjunto S, precisamos escolher um vértice associado a um literal verdadeiro em cada uma das cláusulas.

Como S é um conjunto independente, não há problema de ter valores contraditórios para uma variável, já que vértices associados a variáveis negadas possuem uma aresta entre si em G.

Além disso, como todos os vértices associados a literais de uma mesma cláusula estão ligados, S não possui mais de um vértice associado a uma mesma cláusula. Como $|S| = k$, temos que a valoração definida satisfaz todas as cláusulas de C. Portanto, neste caso, 3-SAT(C) tem resposta *sim*. Logo, 3-SAT $\leqslant_p$ IS.