Exame de Qualificação 11/12/2014

Álvaro Freitas Moreira e Luciana S. Buriol

November 4, 2016

Teoria da Computação

Material

- Turing & The Halting Problem - Computerphile
- Importance and consequences- Wikipedia
- Algorithmics: The Spirit of Computing, p. 228
- Models of Computation and Formal Languages, p. 395

Perguntas

1. **Escreva sobre a tese de Church-Turing. Em sua resposta você deve:**

   (a) **Escrever o enunciado da tese**

   - *Church Turing Thesis*: If function f is effectively calculable, then f is Turing-computable. Equivalently, if a function f is not Turing Computable, then f is not effectively computable.

   (b) **Explicar porque é uma tese e não um teorema (ou seja, explicar por que ela não pode ser provada)**

   - A Tese de Church-Turing se trata de uma tese e não de um teorema, uma vez que não pode ser provada. Se a afirmação indicando que uma função f é efetivamente computável também é Turing-computável fosse falsa, deveria ser esperada a existência de ao menos uma função que pode ser efetivamente computável mas que ao mesmo tempo não é Turing-computável. O fato é que esta função nunca aconteceu ou jamais foi encontrada, sugerindo - mas de nenhuma forma provando - que esta função não existe, que por sua vez significa que a afirmação inicial é verdadeira.
   - O motivo para isso é que entre todos os conceitos que esta tese envolve existe um que é informal e impreciso, chamado de "efetivamente computável" (*effective computability*). A tese iguala a matematicamente precisa noção de "computável por uma máquina de Turing" com a informal, intuitiva noção de "computável efetivamente," que alude que para todos computadores reais e todas as linguagens de programação, aquelas que nós conhecemos no presente e também aquelas que não. Isso soa muito mais como uma especulação selvagem do que o que ela realmente é: uma profunda e abrangente afirmação, colocada por dois dos pioneiros mais respeitados da ciência da computação teórica.

   (c) **Escrever sobre as evidências que suportam a sua verdade (mencionar duas dessas evidências)**

- Por muito tempo pesquisadores tentaram desenvolver um computador dito universal tentando capturar a noção elusiva de efetivamente computável. Turing sugeriu seu modelo de máquinas primitivas e Church desenvolveu um formalismo matemático simples de funções chamadas cálculo lambda. Outros métodos desenvolvidos, como é o caso das funções recursivas, também obtiveram sucesso em utilizar seus modelos para resolver diversos problemas algorítmicos que ficaram conhecidos como algoritmos efetivamente computáveis. Alguns modelos mais complexos se aproximavam até mesmo dos computadores atuais, porém, o fato crucial sobre esses modelos é que todos se mostravam equivalentes em termos de classe da algoritmos que eram capazes resolver.
- Em resumo, o fato que diversas pessoas trabalharem com diferentes ferramentas e técnicas serem capazes de capturar o mesmo conceito serve como evidência para a profundidade da afirmação da tese de Church-Turing. Devido todas elas partirem de um mesmo conceito e terminarem com aparentemente diferentes definições, mas equivalentes, podemos utilizar isso como justificativa para assemelhar essa noção intuitiva com os resultados das outras definições precisas.

(d) **Explicar a sua importância para a Ciência da Computação, em particular explicar porque ela é importante em provas de resultados negativos, como a indecidibilidade do Problema da Parada.**

- Assumindo a tese de Church-Turing, se uma função f não é Turing computável, ela também não é computável em qualquer outro sentido. Isso é, se a tese de CT é verdadeira, então essa função f está estritamente além das fronteiras do que é computável.
- O Problema da Parada é historicamente importante porque foi um dos primeiros problemas à ser provado como indecidível (ambas as provas foram publicadas dentro um curto espaço de tempo por Church e Turing). Subsequentemente, foi possível provar que outros problemas também pertencem a esta classe (problema indecidível); o método típico para provar que um problema é indecidível é a redução. Para fazer isso, é necessário mostrar que se uma solução para um novo problema for encontrada, ela poderá ser utilizada para decidir se um problema é indecidível, seguindo um processo onde as instâncias de um problema indecidível são transformadas em instâncias de um novo problema. Uma vez que já sabemos que nenhum método pode resolver o antigo problema, por consequência, nenhum método também é capaz de solucionar o novo problema. Frequentemente, o novo problema é reduzido para a solução do Problema da Parada.

2. **Faça um breve ensaio discorrendo sobre os seguintes conceitos relativos à Teoria da Complexidade:**

   (a) **Complexidade de tempo de um algoritmo e complexidade de tempo de um problema algoritmico.**

   (b) **Complexidade de tempo polinomial e complexidade de tempo exponencial.**

   (c) **Problemas tratáveis e intratáveis (com exemplos de problemas tratáveis e intratáveis, e exemplos de problemas com "status" em relação a tratabilidade desconhecida).**

3. **A fim de entender melhor problemas algorítmicos com respeito ao uso de recursos, pesquisadores começaram a agrupá-los de acordo com suas semelhanças em relação a requisitos de tempo (e espaço). Esse estudo deu origem as classes de complexidade. Faça um breve ensaio sobre as classes *PTIME, NPTIME, NPTIME-Complete, EXPTIME.* Descreva quais são as propriedades determinantes para um problema estar dentro de cada uma dessas classes. Complemente o ensaio com diagramas de Venn que mostram a relação entre essas classes. Discutir sobre a questão P = NP.**