Algoritmos e Grafos
Mestrado e Doutorado em Ciência a da Comutação

2012

Raimundo Macêdo

ALGORITMOS E GRAFOS
Carga Horária: 34H - Créditos: 02

Ementa:
Conceitos de algoritmos, análise e eficiência de algoritmos, projeto de algoritmos (indução, divisão e conquista, programação dinâmica, método guloso), NP-completude (teoria e técnica de demonstração), classes de complexidade (P, NP, NP-completo, NP-difícil), reduções polinomiais, algoritmos para problemas NP-completos. Conceitos básicos de grafos e algoritmos para resolver problemas modelados em grafos; Conectividade;
Distâncias. Estabilidade e Número Cromático. Árvores e Arborecências; Grafos Planares. Caminhos; Ordenação Topológica; Coloração.

Bibliografia Básica:
•Bondy, J.A., Murty, U.S.R. Graph Theory with Applications. Elsevier Science Publishing, 1976.
•Cormen, T. , C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to algorithms. MIT Press/McGraw Hill, 1990.
•Diestel, Reinhard. Graph Theory. Springer (2000).
•T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Algoritmos: Teoria e prática. Tradução da Segunda edição Americana. Editora Campus, 2002.
•Greenlaw, R. and Hoover, H. J. Fundamentals of the Theory of Computation: Principles and Practice. Morgan Kaufmann, 1998.
•Manber, U. Introduction to Algorithms: A Creative Approach. Addison Wesley, 1989.

•Baase, S., Computer Algorithms. Introduction to Design and Analysis, Second Edition, Addison-Wesley, 1988.

Bibliografia Complementar:
•Parbery, I. Problems on algorithms. Prentice Hall, 1996.
•How to prove it. A Structured Approach. Econd Edition. Daniel J. Valleman
•Martin, J. C. Introduction to Languages and the Theory of Computation. 2nd edition, McGraw Hill, 1997.
•Hopcroft, J. E. and Ullman, J. D. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Pub Co., 1979.
•Goorich, M., Tamassia, R. Projeto de Algoritmos: Fundamentos, análise e exemplos da Internet. Editora Bookman, 2002.
•Ziviane, Nivio. Projeto de Algoritmos: com implementações em C e Pascal. Pioneira Thomson Learning, 1993.
•Sedgewick, R. Algorithms. Second Edition. Addison-Wesley, 1988.
•Sedgewick, R. An Introduction to the Analysis of Algorithms. Addison-Wesley, 1996.
•Skiena, R. S. The Algorithm Design Manual. Springer Telos, 1998.
•Garey, M., David, J. Computers and Intractability: A Guide to the Theory of NP-Completeness W H Freeman & Co., 1979.
•Knuth, D. E. The Art of Computer Programming. Vol. (1-3). Addison-Wesley, 2nd edition, 1997.

Avaliação (intenção apenas)

Participação – 50%
Um seminário sobre um problema NP-Completo - 50%

Freqüência mínima: 75%

Material das aulas em www.macedo.ufba.br

Vocês saberiam dizer a diferença entre computável e tratável?

E a diferença entre P e NP? São de Fato diferentes?

Qual o Significado de $O(N^2)$?

Um pouco de historia

origem dos algoritmos e grafos, maquina de turing, computabilidade ....

Eficiência x Não-Eficiência

Tratável x não tratável

Computável x Não computável

Maquinas de Calcular e Algoritmos existem há muitos séculos

400 a.C. formas primitivas de ábaco (China e Babilonia)
100 a.c. maquina de calcular posições do sol e planetas (Gracia antiga)
.....

1623 - Schuckard (relógio de calcular em Tubingen)
1630 – Oughtred (régua de calculo)
1642 – Pascal (maquina de calcular para números com 8 dígitos)
1673 – Leibniz inventa maquina de calcular (raízes quadradas)
.....

1800-20? Jacquard cria cartões para controlar padrões de tecelagem (programação)
1823 – Babbage (maquinas de diferenças 1) – calculo de tabelas
1854 – Boole (lógica como matemática – lógica binária)
1896 – Hollerith (maquina de leitura de cartões perfurados)

.....

1937 – Turing publica “On computable numbers” noção de maquina universal e Seus limites
1948 – MADAM (Manchester Automatic Digital Machine)
(primeiro computador com programa armazenado) – primeiro programa: fatorar 4620 = (2^2, 3,5,7,11)

Curiosidades !!!!

Como Allan Turing chegou à noção de maquina para processar algoritmo
e a noção de computabilidade ??

Um dado empreendimento foi fundamental

Em 1913 Russel e Whirehead (Cambridge) tentaram estabelecer um fundamento filosófico para a matemática (Principia Mathematica) ... Tentaram mostrar que toda a matemática podia ser derivada de axiomas (lógicos) fundamentais .... Da mesma forma tentou Hilbert (Gottingen). Esse esforço acabou em impasses lógicos.

Exemplo, tomando a frase "o que estou dizendo é falso"
Se a frase for verdadeira, o que ela esta dizendo é falso ... Se for falso, o que diz e verdadeiro.
A proposição e indecidível, portanto.

Mas, muitos acreditavam que essas dificuldades seriam superáveis até que Kurt Gödel, em 1931, mostrou ser impossível provar ser verdadeira a proposição "essa afirmação não
pode ser provada" (levaria a uma contradição) e ser tb impossível provar a falsidade.

Ainda mais, demonstrou que dentro de qquer sistema matemático rigidamente lógico, haverá sempre proposições que não podem ser provadas e tampouco refutadas, com base nos axiomas sobre os quais esse sistema se assenta.

➔A matemática seria, portanto, incompleta ou defeituosa .... Não poderíamos,pois,
ter a certeza que seus axiomas básicos não levariam a contradições.

Nessa época, já era consenso que a matemática poderia produzir proposições arbitrarias
(que não poderiam ser provadas ou refutadas)

Mas a matemática continuava a ser útil .... Nos cálculos, geometria ....

QUESTÃO !!!!

Seria possível determinar que uma proposição seria arbitraria a partir do sistema??

Ou podia uma dessas proposições arbitrarias ser identificada mediante a aplicação de um conjunto de regras derivadas dos axiomas básicos em que o sistemas se fundava?

Poderia isso ser determinado a partir de passos fixos ou procedimentos mecânicos que pudessem ser seguidos por qualquer pessoa ou mesmo uma maquina??

SE SIM, essas proposições arbitrarias poderiam ser identificadas e postas de lado ...

ESSE FOI O PROBLEMA QUE TURING TENTOU RESOLVER !!!

Quais seriam os procedimentos (ou regras) mecânicos que poderiam ser usados para se determinar se uma proposição matemática era ou não suscetível de prova? O que era um número computável e como calculá-lo?

O cálculo seria uma processo rígido que poderia ser seguido por uma máquina.

→Turing definiu então a natureza teórica dessa máquina, que seria capaz de calcular tudo para o que houvesse um algoritmo (i.e., uma seqüência precisa de passos conduzindo a uma conclusão. http://www.turing.org.uk/turing/index.html

Essa máquina universal seria capaz de identificar proposições arbitrárias num sistema Matemático … Ela também poderia ser alimentado com um número que codificasse a descrição de uma outra máquina de Turing, e em seguida se comportar como esta (calculando Primos, jogando xadrez, etc.)

QUESTÃO: Mas o que aconteceria se essa máquina fosse alimentada com o número que codificava sua própria descrição???

Como ela iria se comportar como si mesma? E Como poderia seguir o procedimento que a fazia se comportar como si mesma quando já estava se comportando como si mesma ….. Seria uma auto-contradição …..um calculo não computável.

Seria impossível elaborar um conjunto de regras capaz de resolver se uma proposição era suscetível de prova ou refutação usando unicamente os procedimentos oriundos daquele sistema (Similar a prova de Güedel)

Conclusão

Computadores podem computar muitas coisas (para os quais existam algoritmos), mas algumas coisas são incomputáveis (o problema da parada) !!!!

---------- Era COOK (The Complexity of Theorem-Proving Procedures - Stephen A. Cook, University of Toronto – 1971)

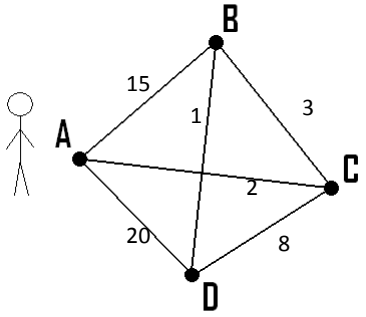
Mas será que todo problema para o qual existe um algoritmo existe de fato uma solução prática?

Considere esse exemplo: calcule n^m , onde n e m são números naturais quaisquer.

Existirão instâncias desses problema cujas soluções vão contra os princípios naturais:
Ex: a matéria no "universo" pode ser menor do que o espaço necessário para escrever O resultado.

Outros problemas de maior interesse prático sofrem da mesma dificuldade.
Vejamos a seguir.

"Given the distances between any two points, what is the shortest route a salesman can make from point A, visiting all points, and returning to point A? This is the Travelling Salesman problem, an NP-Hard problem in mathematics."



Dado um mapa genérico !!!!
Não existe uma solução eficiente para todas as instâncias desse problema.
É um problema de otimização combinatória: minimizar uma somatória.

Suponhamos temos um computador muito veloz , capaz de fazer 1 bilhão de adições por segundo

No caso de 20 cidades, o computador precisa apenas de 19 adições para dizer qual o comprimento de uma rota e então será capaz de calcular 10^9 / 19 = 53 milhões de rotas por segundo. Contudo, essa imensa velocidade é um nada frente à imensidão do número 19! de rotas que precisará examinar. O valor de 19! é 121 645 100 408 832 000 (ou , aproximadamente, 1.2 x 10^{17} em notação científica). Conseqüentemente, ele precisará de 1.2 x 10^{17} / (53 milhões) = 2.3 x 10^9 segundos para completar sua tarefa, o que equivale a cerca de **73 anos ...**

O problema é que **a quantidade (n - 1)! cresce com uma velocidade alarmante, sendo que muito rapidamente o computador torna-se incapaz de executar o que lhe pedimos.**

n	rotas por segundo	(n - 1)!	cálculo total
5	250 milhoes	24	insignific
10	110 milhoes	362 880	0.003 seg
15	71 milhoes	87 bilhoes	20 min
20	53 milhoes	1.2 x 10^{17}	73 anos
25	42 milhoes	6.2 x 10^{23}	470 milhoes de anos

Origem da tabela: http://www.mat.ufrgs.br/

Uma aplicação prática …

Uma fábrica de componentes eletrônicos precisa soldar milhares de placas de circuitos impressos. Para agilizar, todas as placas de mesmo tipo são soldadas uma após a outra, em pontos previamente especificados. O objetivo é saber qual o trajeto das posições que a máquina de solda deve percorrer para que gaste o menor tempo possível em cada placa. Note que qualquer ganho de tempo neste trajeto pode ser crucial para se soldar as milhares de placas em pouco tempo.

A função de complexidade de tempo de um algoritmo expressa a maior quantidade de tempo necessária para o algoritmo resolver o problema para qualquer entrada possível de um dado tamanho.

Algoritmos distintos possuem funções distintas de complexidade de tempo. Em geral, se distinguem em
algoritmos eficientes e não eficientes, que podem ser caracterizadas pelo seu aspecto polinomial ou exponencial, respectivamente.

Para a definição da função é necessário a definição do que seria o tamanho do problema. Ex: no caso do problema de ordenação de vetores ou tabelas, o numero de entradas do vetor ou tabela e utilizado.

Uma definição mais formal

f(n) = O(g(n)), se existe constante *c* t.q. $|f(n)| \leq c. |g(n)|$ para todo $n \geq 0$

Ou $\lim_{x \to x0} f(x)/g(x) \to K$

A complexidade é polinomial se definida como O(p(n)) para alguma função polinomial p, onde n e o tamanho do problema.

Se a função de complexidade não puder ser limitada por polinômio, diz-se que a mesma possui complexidade exponencial.

Edmonds (1965) considerava algoritmos de tempo polinomial como algoritmos "bons". E conjeturou que certos problemas de programação inteira poderiam não possuir soluções boas (ou polinomiais)

The Complexity of Theorem-Proving Procedures - Stephen Cook, University of Toronto – 1971)

Teoria dos problemas NP-COMPLETO

The Complexity of Theorem-Proving Procedures
Stephen A. Cook
University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

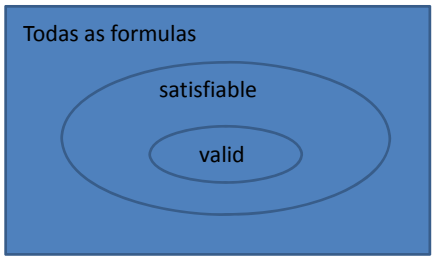
Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

Palavras Chaves

Recognition problem: problema de decisão

"polynomial timebounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology".

Calculo proposicional



Deif1:

Uma formula proposicional A é satisfiable se alguma interpretação para A (associação de valores T ou F) leva a Verdade.
A é valida se todas as interpretações possíveis levam a Verdade (tautologia).

Def2:

A é unsatisfiable ou contraditória se não é satisfiable (i.e., todas as interpretações de A levam a Falso)
A é náo-valida se alguma interpretação leva a Falso

Teorema 1

A é valida sse não-A é unsatisfiable (não satisfativel). A is satisfiable sse não-A é náo-valida

Pelo teorema 1, um procedimento de decisão para satisfiability pode ser usado para construir um procedimento de decisão para validade.

Para decidir se A é valido, aplica-se o procedimento de decisão para a satisfiability de não-A

Teorema 1

A é valida sse não-A é unsatisfiable. A is satisfiable sse não-A é náo-valida

Pelo teorema 1, um procedimento de decisão para satisfiability pode ser usado para construir um procedimento de decisão para validade.

Para decidir se A é valido, aplica-se o procedimento de decisão para a satisfiability de não-A

Ou seja, se A é válido (todos as interpretações são T), não-A deve ter todos as interpretações como F. Portanto, se não-A é satistiable (existe uma interpretação T), A não pode ser válida (pois tem pelo menos uma interpretação F)

E para calcular satisfiability para o calculo proposicional??

OBS: existem 2^n interpretações possíveis
FORÇA BRUTA: Tabela de verdade Ex: A = (p →q) → (¬q → ¬P) é válida

p	q	p →q	¬q → ¬P	= (P →q) → (¬q → ¬P)
T	T	T	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	T	T

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

Não válida
Satisfiable

Para a formula ¬p ^ ¬q ^ ¬r , a satisfiabilidade somente é verificada na última insterpretação

p = q = r = F ….. a única interpretação que leva a T (são $2^3 = 8$ interpretações)

Pergunta: existe um método satisfiability mais substancialmente eficiente que tabela verdade?

Resp: NÃO (ou não se sabe)

P = NP?

Imagine agora uma máquina não-deterministica, que pode avaliar tantas interpretações quanto necessárias de forma eficiente (em paralelo) …. Palalelismo infinito.

Dada uma formula A. Escolha uma interpretação v tq v(A) = T.

➔Nesse caso, a verificação de satisfabilidade de uma interpretação é feita em tempo polinomial usando uma máquina (ou algoritmo) não deterministico. Portanto, Satisfiability está na classe NP (nondeterministic polynomial) (prova de Cook)

Não se sabe se tal poblema pode ser resolvido em tempo polinomial por uma algoritmo deterministico. Ou seja, se P = NP?

A partir desse trabalho, vários autores provaram problemas que estão em NP

Muitos exemplos em Grafos ….

Conclusão ….

A problem is called NP (nondeterministic polynomial) if its solution can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. If a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-complete. Thus, finding an efficient algorithm for any NP-complete problem implies that an efficient...

Cook escreveu:
It is shown that any recognition problem solved by a polynomial time bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology.

O sentido de redução que ele usa é o seguinte:

Qualquer instancia do problema " recognition problem solved by a polynomial time bounded nondeterministic Turing machine“ pode ser reduzido a uma instância de "determining whether a given propositional formula is a tautology", se qualquer dessas instancias do primeiro problema pode ser formulado numa instância correspondente do segundo problema. Uma solução para a instância do segundo problema irá resolver a instância correspondente do primeiro problema. Exemplo hipotético: resolver x*x*x é reduzido em resolver x*y para toda instancia de x*x*x

Portanto se eu tenho uma solução para x*y, eu sei que posso resolver x*x*x

Nesse caso “any recognition problem” não é mais difícil de resolver do que satisfiability.
(ele pode ser até mais fácil) E uma solução para satisfiability leva a solução do outro problema.

Nesse enunciado, any recognition problem forma a classe dos NP-Completo

Essencialmente, essa foi a prova de Cook de que satisfiability é NP-COMPLETO

Num outro teorema, Cook diz:

Theorem 2: The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D3, {subgraph pairs}.

Isso se usa para verificar se um problema é NP-Completo. ... Fazendo a redução de um NP-COMPLETO já conhecido para outro NP que deseja incluir na classe NP-Completo

PS: veja o comentário de Cook para o teorema 2

Remark: We have not been able to add {primes} or {isomorphic graph pairs} to the above list. To show (tautologies} is P-reducible to (primes} would seem to require some deep results in number theory, while showing {tautologies} is P-reducible to (isomorphic graph pairs} would probably upset a conjecture of Corneil's [4] from which he deduces that the graph isomor- . phism problem can be solved in polynomial time.

Mas, enfim o que significa redução?

Redução e Transformação é a mesma coisa

Um problema Q pode ser reduzido ao um problema Q' se qualquer instância de Q pode ser facilmente expresso numa instância de Q' de modo que uma solução para este último (instância de Q') representa uma solução para a instância correspondente de Q.

Como identificar que um problema Q é NP- Completo??

1) Mostra-se que Q está em NP
2) Seleciona-se um problema NP - Completo conhecido Q' (ex: satisfiability)
3) Constrói-se uma transformação ou redução f de Q´ para Q
4) Prova-se que f é uma transformação polinomial

Se somente (1) não pode ser satisfeito, o problema é chamado de NP-HARD.

Nem sempre é possível se determinar que o problema é NP. Ex: considere o complemento do problema do caixeiro viajante: dado um conjunto de cidades e suas distâncias, é verdade que nenhum percurso entre as cidades tem tamanho B ou menor que B?

Não é possível dar um resultado "sim" sem percorrer todos ou a maioria dos percursos.

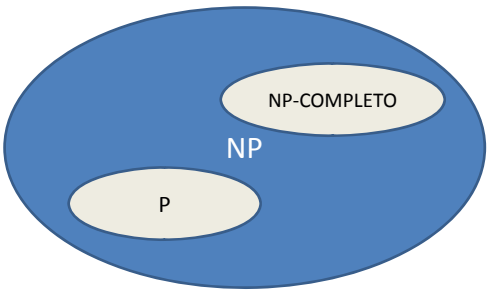


Figura RESUMO

Para a próximo aula trazer um resumo de máximo uma página sobre o que foi mostrado na aula do dia 9/03/12.

Um aluno será escolhido por sorteio para falar sobre seu resumo e o que fora aprendido na aula