THEORY OF COMPUTATION

Syllabus for the course CS 5315/6315, Spring 2018

Instructor: Vladik Kreinovich, email vladik@utep.edu, office CCSB 3.0404, office phone (915) 747-6951

Class time: TR 4:30-5:50 pm, Room CCSB 1.0202

Instructor: Vladik Kreinovich, email vladik@utep.edu, office CCSB 1.0510, office phone (915) 747-6951.

- The instructor's office hours are: Tuesdays and Thursdays 2-3 pm and 6-7 pm, or by appointment.
- If you want to come during the scheduled office hours, there is no need to schedule an appointment.
- If you cannot come during the instructor's scheduled office hours, please schedule an appointment in the following way:
  - use the instructor's appointments page http://www.cs.utep.edu/vladik/appointments.html to find the time when the instructor is not busy (i.e., when he has no other appointments), and
  - send him an email, to vladik@utep.edu, indicating the day and time that you would like to meet.

  He will then send a reply email, usually confirming that he is available at this time, and he will place the meeting with you on his schedule.

Prerequisite: CS 3350 (Automata)

Main Objectives:

- to teach the foundations of computing, and
- to enable students to prove (not only to program).

Contents

1. *Turing's snakelike machine:* not very fancy, but it can compute anything (you just wait and wait and wait, ...). Finally: something purely theoretical (and not real machines): *recursive functions*. Church's bold statement: if anyone can compute anything on any machine, I can compute it on a Turing snake! Universal Turing machine. Can anyone really beat Church? We'll discuss the attempts (Gandi, Kreisel, etc) if time allows.
2. You are accustomed to the fact that everything is computable, and if your program does not work, that means a bad grade. Finally! Only in this course! Computational problems that cannot be solved! (and so you get a bad grade, if your program solves them - just kidding).
3. If a program requires a billion years to finish its computations, only a crazy theoretician can call it an algorithm. So, to sound more reasonable, we will talk about computational complexity, realistic (polynomial-time) computations, P and NP, NC, limitations on space and on the number of processors, etc. "P=NP?" as a challenge to mankind. Will science ever stop? Again, we will find here lots of undecidability results. And maybe, as a project, you will be able to prove that some problem that you were planning to solve is undecidable.

4. What to do if your problem turned out to be undecidable? For sure: not to give up. It can be still decidable in some sense: for almost all cases, by a Monte-Carlo algorithm that gives an answer with probability close to 1, etc. Few results and lots of open problems.
5. Turing machine was invented in the 30s, P=NP problem appeared when many of you guys were too young to count. What is the modern state of the Theory of Computation? We'll try to cover briefly:
    - parallelism,
    - algebraic computations,
    - computations with real data (including the idea of interval mathematics),
    - fuzzy algorithms,
    - neural computing,
    - chemical computing,
    - quantum computations, etc.

Learning Outcomes

1. **Knowledge and Comprehension**

a. Describe the practical need for theory of computing: to know which computational problems are solvable and which are not, and which problems can be, in principle, solved within given computation time, to avoid wasting resources on trying to solve problems in such a general context that they become unsolvable

b. Describe different models of computing, including recursive functions, different versions of Turing machine, etc.

c. Describe how computability in a programming language is related to formal models of computing, on the example of primitive recursive and recursive functions

d. Understand Church-Turing thesis and understand the status of this thesis - that it is, in effect, a statement about the physical world

e. Define decidable and recursively enumerable (r.e., semi-decidable) sets

f. Understand the notion of an oracle and of computing relative to an oracle

g. Define classes P, NP, the notions of polynomial time reduction, NP-hardness, and NP-completeness; understand the motivations behind these definitions: P means feasible, NP stands for a problem; understand the difference between the formal notion of polynomial time and the practical notion of feasibility

i. Understand the difference between a proof and a sequence of reasonable arguments which does not constitute a proof

j. Know several NP-hard problems

k. Define complexity classes from the absolute and relative polynomial hierarchy; give examples of problems

l. Understand the main idea of parallelization

m. Define the class NC of parallelizable P problems, know the relation between NC and P, and an example of a P-complete problem

n. Understand the difference between the formal computation time of a parallel program and the actual time which takes communication time into account

o. Define Kolmogorov complexity, understand motivations behind this definition

p. Be aware of the main ideas behind different physical schemes of computing beyond traditional Turing machines, such as quantum computing

2. **Application and Analysis**

a. Trace the computation of a primitive recursive or recursive function on a numerical example

b. Translate a formal description of a primitive recursive or recursive function into a program

c. Trace a Turing machine on a given input

d. Prove that satisfiability is NP-hard, and that one more problem is NP-hard

e. Know how to parallelize standard simple parallelizable algorithms (e.g., search, or dot product of two vectors)

f. Use the definition of Kolmogorov complexity K(x) to provide reasonable upper bounds for K(x) of a given string x

3. **Synthesis and Evaluation**

a. Prove, from the definition, that a given function (e.g., a given polynomial or propositional function) is primitive recursive or recursive

b. Design a Turing machine that computes a given function

c. Synthesize Turing machines that compute two functions into a Turing machine for computing their composition

d. Prove that not every problem is computable - e.g., that the halting problem is undecidable

e. Apply diagonalization to prove results similar to what we had in class

f. Prove that given simple sets are decidable and/or r.e.

g. Prove that the union, intersection, and complement of decidable sets are decidable

h. Prove that the union and intersection of r.e. sets is r.e.; prove that the complement to a r.e. set is not always r.e.

i. Prove that not every r.e. set is decidable and that not every set is r.e.

j. Prove, for a given software requirement, it is algorithmically impossible to check whether a given program satisfies this requirement

k. For problems similar to ones considered in class, prove their NP-hardness

l. Be able to parallelize simple algorithms

m. Be able to understand and present a research paper in Theory of Computing - with a minor help from a professor

Main Source: Michael Sipser, Introduction to the Theory of Computation, PWS Publishing Co., 2nd or later edition

Projects: An important part of the class is a project. There are three possible types of projects:

- An ideal class project is if you do something related to theory which is useful for your future thesis or dissertation. Please check with your advisor about it, maybe he or she wants you to read and report on some theory-related paper, maybe you need to do some theory-related research, whatever your advisor recommends will be a very good project for this class, just let the class instructor know what exactly you plan to do.
- If you have not yet selected an advisor, but you already know what research area you want to work in, come talks to the class instructor, we will try to find some appropriate topic -- and if you have any proposals already, great.
- If you do not have a research topic or you have a one but your advisor cannot find anything theory-related that will be helpful for your future thesis or dissertation, come talk to the instructor too.
- Maybe you like theory and want to start doing a project in theory, then come and talk to the instructor, we will try to find something that will be of interest to you.

A project can be:

- reviewing and reporting on a related paper, or
- doing some independent research (not research as in high school, but research as in graduate school, i.e., trying to come up with something new), or
- programming something theory-related.

The most important aspect of the project is that it should be useful and/or interesting to *you*. The instructor can assign a project to you, there are plenty of potential projects, but if each student selects a project that he or she likes, this will be much much better for everyone.

Tests and Grades: There will be three tests and a final exam. Each topic means home assignments (mainly on the sheets of paper, but some on the real computer). Some of them may be graded. Maximum number of points:

- first test: 10
- second test: 10
- third test: 15
- home assignments: 10
- final exam: 35
- project: 20

(smart projects with ideas that can turn into a serious scientific publication get up to 40 points).

A good project can help but it cannot completely cover possible deficiencies of knowledge as shown on the test and on the homeworks. In general, up to 80 points come from tests and home assignments. So:

- to get an A, you must gain, on all the tests and home assignments, at least 90% of the possible amount of points (i.e., at least 72), and also at least 90 points overall;
- to get a B, you must gain, on all the tests and home assignments, at least 80% of the possible amount of points (i.e., at least 64), and also at least 80 points overall;
- to get a C, you must gain, on all the tests and home assignments, at least 70% of the possible amount of points (i.e., at least 56), and also at least 70 points overall.

Standards of Conduct: You are expected to conduct yourself in a professional and courteous manner, as prescribed by the UTEP Standards of Conduct.

Graded work, e.g., homework and tests, is to be completed independently and should be unmistakably your own work (or, in the case of group work, your team's work), although you may discuss your project with other students in a general way. You may not represent as your own work material that is transcribed or copied from another person, book, or any other source, e.g., a web page.

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion.

- Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports.
- Plagiarism occurs when someone represents the work or ideas of another person as his/her own.
- Collusion involves collaborating with another person to commit an academically dishonest act.

Professors are required to -- and will -- report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

Disabilities: If you feel you may have a disability that requires accommodation, contact the The Center for Accommodations and Support Services (CASS) at 747-5148, go to Room 106 E. Union, or e-mail to cass@utep.edu. For additional information, please visit the CASS website.

See You All in the Class!