CSC 373 Lecture 16

- Announcements: Regrading policy; question 2 of term test 1; 5 questions on new assignment; TA office hour?
- Review: Any questions regarding flow networks?
- New topic NP sets (decision problems) and NP completeness. Read chapter 8; you can just initially skim sections 8.5,8.6,8.7
- Motivation, Polynomial time, Polynomial time reduction, Some simple reductions
- Mainly a board talk to slow down

Motivation

- One of the perhaps greatest ideas in computing is that we basically all agree on that it means for a (discrete) computational problem to be computable; namely, we equate the intuitive concept of "computable" with the mathematically precisely defined concept of Turing computable. This is the so-called Church-Turing thesis (from ~1936) and although one can never prove such a hypothesis, it is almost universally believed. Why?
- One can use diagonalization to show that for any time bound *T(n)*, there are computable functions not computable within time *T(n)*. So what does it mean to be "efficiently computable" ?

Efficient Church Turing thesis

- We will equate the intuitive concept of “efficiently computable” with computable in polynomial time (i.e. time bounded by a polynomial function of the encoded length of the inputs and outputs.
- This has sometimes been called Cook’s Thesis. This hypothesis is not literally believed (why?) but it is an abstraction that has led to great progress in computing.
- Informal claim: Any function (poly time) computable is (poly time) computable by a Turing machine. For the time being we will not introduce a precise computational model.

Now for some formalities

- We are always assuming that inputs are encoded as strings over some finite alphabet S with at least 2 symbols. (We can use as many symbols as we want but 2 suffices for our purpose. Note: finitely many symbols on a keyboard.)
- We can encode a set of inputs *w_1, …w_n* by having a special symbol (say #) to separate the inputs but again this can all be encoded back into 2 symbols.
- We say that a function *f:S** into *S** is computable in time T() if there is an algorithm (to be precise a Turing machine or an idealized RAM with an appropriate instruction set) such that for all inputs w, the algorithm halts using at most *T(n)* steps where $n = |w| + |f(w)|$. We will never be dealing with functions where $|f(w)| >> |w|$ so *n* will then just be the length of the input.

Some definitions and notation (mainly on the board)

- A polynomial time reduction (called a polynomial time Turing reduction)
- A polynomial time transformation - special case of a poly time reduction (called a “many to one poly time reduction)
- Simple observation we already made : If problem X poly time reduces to problem Y, then if Y is computable in poly time then so is X. The contrapositive is that if X is not poly time computable then Y is not poly time computable.
- Note: poly time reduction and transformation are transitive relations.

Some relatively easy transformations

- Vertex cover transforms to independent set and conversely, independent set transforms to vertex cover.
- Note: these are NP complete problems and all such problems can theoretically be reduced to each other. But here the reduction in both directions is immediate.
- SAT to 3-SAT (Clearly here the converse holds.)
- 3-SAT to Clique