Algorithms and Data Structures

Week 13 – Lecture A

Computational Complexity

- All problems can be classified into one of a number of classes, depending on how difficult they are to solve.
- These classes include:
  - P – the set of all problems which can be solved in *Polynomial* time;
  - EXP – the set of all problems which can be solved in *Exponential* time;
  - R – the set of all problems which can be solved in *Finite* time.
- Beyond R there are still problems, these are *unsolvable*.
  - There are more of these than in any other class. ☹
- Almost all of the problems we have looked at in this subject are members of P.

Some Sample Problems

- Negative-Weight Cycle Detection:
  - $\in$ P.
- $n \times n$ Chess:
  - $\in$ EXP;
  - $\notin$ P.
- Tetris:
  - $\in$ EXP;
  - We don’t know if it is in P.
- Halting Problem:
  - $\notin$ R.

Most Decision Problems $\notin$ R

- Decision problems are ones with a Yes/No answer.
- Program
  - ⇨ Binary string
  - ⇨ Natural number.
- So there are no more programs than there are integers.
- Decision Problem
  - ⇨ function: inputs $\rightarrow$ {Yes, No}
  - ⇨ function: binary string $\rightarrow$ {0,1}
  - ⇨ function: natural number $\rightarrow$ {0,1}.

Most Decision Problems $\notin$ R

- We can tabulate any decision problem for each of its possible inputs…

Input	1	2	3	4	5	6	7	8	…
Yes/No	0	1	0	0	1	1	0	1	…

- We can express any decision problem in terms of the infinite sequence of bits in the Yes/No row.
- If we express it as a binary fraction – e.g. .01001101… we can equate each decision problem to a real number between 0 and 1.

Most Decision Problems $\notin$ R

- So:
  - Every program $\in \mathcal{Z}$;
    - $\mathcal{Z}$ is *countably* infinite.
  - Every decision problem $\in \Re$.
    - $\Re$ is *uncountably* infinite.
- $|\Re| >> |\mathcal{Z}|$.
  - There are far more real numbers than integers;
  - There are far more decision problems than programs;
  - Almost every problem is non-computable;
  - Almost every problem is unsolvable by any program.
- Depressed?

One More Class?

- NP – the set of all problems solvable in polynomial time using a "lucky" algorithm.
  - Lucky = always makes the right guess.
- Nondeterministic model:
  - Algorithm makes guesses;
  - Guess leads to Yes or No;
  - Guesses will always lead to Yes if there is at least one solution in the decision tree;
  - This is a sort of "magic greedy algorithm".

Tetris

- If we play a finite game of Tetris we can make guesses for each piece:
  - Where to drop it;
  - How to rotate it;
  - Whether to make last-minute adjustments.
- If we have a "lucky" algorithm for Tetris we can answer the question "Can I survive?" in a number of guesses which is polynomial in the number of pieces.
- This means that Tetris $\in$ NP.

NP – an Alternative View

- NP – the set of all problems whose "solutions" can be "checked" in polynomial time.
  - Whenever the answer is Yes there exists a proof which can be verified in polynomial time.
- It is usually easier to check a solution than to produce it.
  - E.g. Is 54727067 a composite number? – hard to test.
  - The factors of 54727067 are 6701 and 8167. – easy to verify.
- It should be obvious that every problem in P is also in NP.
  - $P \subseteq NP$
  - P = NP ? (Probably not.)

Tetris Again

- If P≠NP we can prove that some problems, including Tetris, are in NP−P.
- We do this by demonstrating that such problems are as hard as possible while still being in NP.
- We can demonstrate that Tetris is NP-hard;
  - At least as hard as every problem in NP.
- In fact, Tetris is NP-complete.
  - NP-hard and in NP.

Reduction: Defining “As Hard As”

- Given a problem, A, that you want to solve.
- If we can convert it into some other problem, B, then we can solve B and use this solution to solve A.
- This process is called *reduction*.
- We say the problem A is *reducible* to problem B.
- Note: this may not be the best way to solve problem A.
- If A is reducible to B we can say that B is at least as hard as A.

An Example of Reduction

- A:
  - Given a weighted graph, G, find the path between S and D which minimizes the *product* of the path weights.
- B:
  - Given a weighted graph, G′, find the path between S and D which minimizes the *sum* of the path weights.
- Reduction A$\rightarrow$B:
  - Replace the weights in G with their logarithms.

Proving a Problem is NP-complete

- To prove a problem is NP-complete we need only to prove:
  - The problem is in NP;
  - Some other problem that is already known to be NP-complete is reducible to the new problem.
- This avoids the issue of finding the first NP-complete problem.
- Stephen Cook did this in 1971:
  - Formula Satisfiability (Sat).
- Richard Karp extended Cook's work:
  - In fact, he showed that 21 different problems were all NP-complete.

Some other NP-complete Problems

- The knapsack problem:
  - Given n elements, $\{w_1, ..., w_n\}$ and a target W, is there a subset of elements which adds up exactly to W?
- The Hamiltonian cycle problem:
  - Given a graph, G=(V, E), is there a sequence of distinct edges $e \in E$ forming a closed path such that each vertex $v \in V$ is visited exactly once?
- The three colour problem.
  - Given a graph, G=(V, E), can colours be assigned to its vertices such that for any two vertices v_1, v_2 such that:
    - if $(v_1, v_2) \in E$ then $colour(v_1) \neq colour(v_2)$ using no more than 3 colours.

Some other NP-complete Problems

- The travelling salesman problem:
  - Given a weighted graph, G=(V, E, W), is there a sequence of distinct edges $e \in E$ forming a closed path such that each vertex $v \in V$ is visited exactly once and the sum of the weights of the traversed edges is minimized?
- Satisfiability:
  - Given a set of Boolean variables $b_1, b_2, \ldots, b_n$ and the operators AND, OR and NOT can values be assigned to each variable such that a well formed expression involving only these components such that the expression evaluates to true?
  - A version of this was the first problem shown to be NP-complete.

Who Wants to be a Millionaire?

- You can win $1,000,000.
- All you have to do is either:
  - Find a polynomial time algorithm to solve any NP-Complete problem, (your choice of problem), or;
  - Prove that at least one NP-complete problem has no polynomial time solution algorithm.
- Note: “I can’t find one” ≠ “There isn’t one”