Turing Machines, part IV

The Definition of Algorithm

In last lecture, we saw several variants of TMs

- Multi-Tape TMs
- Nondeterministic TMs
- Enumerators

And established that they were equivalent to standard TM

Equivalence with yet other models

- Many other models of general purpose computation have been proposed
- Some are very much like TMs, but others are quite different
- All share the essential feature of TMs – namely, unrestricted access to unlimited memory – distinguishing them from weaker models such as FA
- Remarkably, all models with that feature turn out to be equivalent in power, so long as they satisfy reasonable requirements (e.g. the ability to perform only a finite amount of work in a single step)
- This phenomenon is analogous to “equivalence of programming languages”
- This analogy has profound implication – definition of algorithm -- the subject of today’s lecture

What is an algorithm?

- Intuitively, an algorithm is a collection of simple instructions for carrying out some task
- The notion is much much older than Computer Science
  - Gaussian method (18^{th} century)
  - Euclid's method for greatest common divisor (c. 300 BC)
  - Finding prime numbers
- The intuitive notion is insufficient for gaining a deeper understating of algorithms – precise definition had to wait until the 20^{th} century

Carl Fredich Gauss (1777—1855)

Eukleides of Alexandrai (mid-4^{th} century BC – mid-3^{rd} century BC)

What is an algorithm?

- Hilbert's problems
  - In 1900, Hilbert posed 23 mathematical problems as challenges for the coming century
  - His 10th problem was to devise a procedure (an algorithm) that tests whether a polynomial has an integral root
  - Example of polynomial:

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

Portrait of David Hilbert in the 1900s

- We know today *no algorithm exists for this task*
- The intuitive understanding of algorithms (that the 19th century mathematicians had at the time) was insufficient for showing such impossibility result
- Proving impossibility (negative) result requires precise definition of algorithm

What is an algorithm?

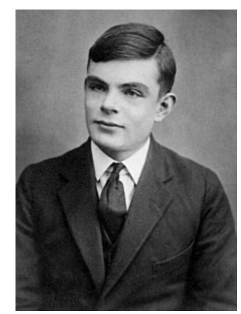
- Hilbert's problems
  - In 1900, Hilbert posed 23 mathematical problems as challenges for the coming century
  - His 10th problem was to devise a procedure (an algorithm) that tests whether a polynomial has an integral root
- We know today *no algorithm exists for this task*
- The intuitive understanding of algorithms (that the 19th century mathematicians had at the time) was insufficient for showing such impossibility result
- Proving impossibility (negative) result requires precise definition of algorithm
- That definition came in 1936 – Church-Turing thesis, which basically states that

Intuitive notion of algorithms

equals

Turing machine algorithms

Portrait of David Hilbert in the 1900s



Alan Turing (1912--1954)

Turing Machines

Alonzo Church (1903--1995)

λ-calculus

Digging deeper into Hilbert's 10^{th} problem

- Let us phrase the problem in our terminology

  Let D = {p | p is a polynomial with an integral root}

- Hilbert in essence asked whether D is decidable
- The answer is negative
- In contrast, we show that it is Turning-recognizable

- Before doing so, let us first consider a simpler problem

Let D_1 = {p | p is a polynomial over a single variable x with an integral root}

Digging deeper into Hilbert's 10th problem

- Here is a TM M_1 that recognizes D_1

M_1 = "on input <p>: where p is a polynomial over the variable x.

  - Evaluate p with x set successively to the values 0,1,-1,2,-2,3,-3,…. If at any point the polynomial evaluates to zero, *accept*."

- If p has an integral root, M_1 eventually will find it and accept. If p does not have an integral root, M_1 will run forever.
- For the multivariable case, we can present a similar TM M that recognizes D.

- Both M_1 and M are recognizers but not deciders.
- We can convert M_1 to be decider for D_1 because we can calculate *bounds* within which the roots of a single variable polynomial must lie and restrict the search to these bounds.
- There is a result (Matijasevic, 1970) that shows that calculating such bounds for multivariable polynomials is impossible.

$$\boxed{\pm k \frac{c_{\max}}{c_1}}$$

Turning point

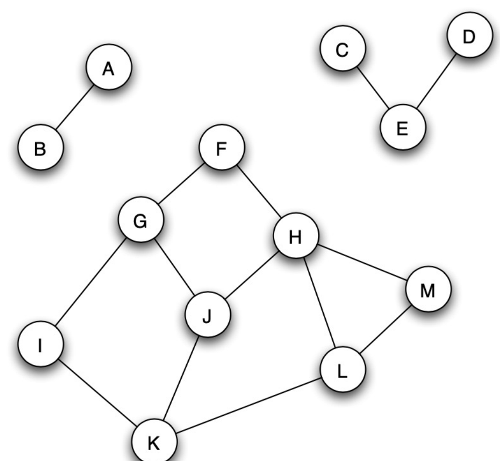
- We have come to a turning point in our study of theory of computation
- We will continue to speak of Turing machines, but our real focus from now on is on algorithms
- TM merely serves as a precise model for the definition of algorithms
- We will rely on high-level description of TM algorithms and follow a certain format and notation
  - The input to a TM is always a string
  - If we want to provide an input other than a string as input, we first represent – encode -- that object as a string. This is always possible.
  - Our notation for encoding an object O into its representation as a string is <O>
  - If we have several objects $O_1, O_2, \ldots, O_k$, we denote their encodings into a single string $\langle O_1, O_2, O_3, \ldots, O_k \rangle$
  - The encoding can be done in many reasonable ways, and it doesn't matter which one we pick.
  - TM algorithms will be described as indented segments of text within quotes.
  - The algorithms are broken into stages, each potentially involving many steps.

Example

- Let A be the language consisting of all strings representing undirected graphs that are connected. That is,

  A = {<G> | G is a connected undirected graph}

- The following (next slide) is a high-level description of a TM M that decides A



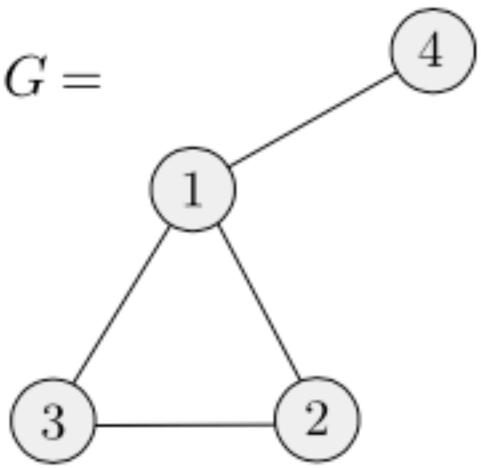
TM that decides A

$M =$ "On input $\langle G \rangle$, the encoding of a graph G:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of G to determine whether they all are marked. If they are, *accept*; otherwise, *reject*."

A graph G and its encoding <G>

$G =$



$\langle G \rangle =$

(1,2,3,4)((1,2),(2,3),(3,1),(1,4))