Computability & Complexity

Scenario

Better scenario

Another possible scenario

Limitations of computer science

What problems can be solved by computers?

Major reasons useful calculations cannot be done:

- Execution time of program is too long
  - problem can take years or centuries to finish
- Problem is not computable
  - no computer program can solve the problem
- We do not know how to write a program to solve the problem
  - problems that can conceivably be solved
    - includes many problems in artificial intelligence and computer vision, such as understanding language, object recognition, tracking, prediction

- Equivalent to: “What decision problems can be solved?”
- Decision problem
  - problem where the answer is always YES/NO
- Arbitrary problem can always be reduced to a decision problem
- There are decision problems for which algorithms do not exist
  - there are more different decision problems than there are different programs
- If there are more decision problems than programs, then it must be the case that some decision problems cannot be solved by programs
### Function evaluation

- A decision problem is a question with a YES/NO answer:
  - Is 613511 a prime number?
  - Is my birthday on a Sunday next year?
  - Did I pass the CPS1 final exam?
- Sometimes a particular answer from a set of possible solutions is required:
  - What is the smallest prime factor of 613511?
  - On what day of the week is my birthday next year?
  - What grade will I get on the CPS1 final exam?
- The process of answering questions where a particular answer is uniquely determined from the given information can be viewed as the evaluation of a function.

### Functions

- A function is a correspondence between a collection of possible input values and a collection of output values such that each possible input is assigned a unique output.
- Example: the addition function
  - inputs are value pairs, outputs are values representing the sum of each input pair
  - output is unique, exactly one value for each input pair
  - Process of determining output for given input is called computing or evaluating the function
  - The ability to evaluate functions gives us the ability to solve problems
  - to solve the addition problem we must evaluate the addition function

### Computable functions

- Algorithms are the means by which we formally express the evaluation of a function.
- Can we design algorithms to evaluate all functions?
  - No!
- Exist functions so complex that there is no well-defined step-by-step process for determining their output based on their input values.
- The evaluation of these functions lies beyond the capabilities of any algorithmic system.
- These functions are said to be noncomputable.
- Functions whose output values can be determined algorithmically from their input values are said to be computable:
  - functions having an algorithm that solves them.

### Limitations of machines

- Machines can only perform computations described by algorithms.
  - the study of computable functions is the study of the ultimate capabilities of machines.
  - If we can build a machine with the capabilities to evaluate the entire set of computable functions, then the machines we build are as powerful as we can make them.
  - On the other hand …
    - if we discover that the solutions to a problem requires the evaluation of a noncomputable function, we know the solution to that problem lies beyond the capabilities of machines.

### “Does extraterrestrial life exist?”

- **Q:** Is it possible to write a Java program to answer the following question: *Does extraterrestrial life exist?*
- **A:** Yes!

- The question is answered either by a program that returns “yes” or by a program that returns “no.
  - We just don’t know which program is correct.

### Is a problem computable?

- Suppose we create a list of all programs for evaluating computable problems.
- To ask whether a problem is computable is to ask if there is a program in the list to solve the problem.
- This is not the same as actually identifying the program in the list that solves the problem.
- There are many computable problems whose solutions have not yet been identified.
Abstract model of computation

- The first abstract model of computation was defined by Alan Turing in 1936
- The Turing machine
- A Turing machine is a purely theoretical device, developed well before technology was capable of providing the machines we know as "computers" today
- Model to study the capabilities and limitations of computational processes and computing machinery
- Provide a context in which solutions to all things computable can be expressed
- Provide a standard to which the power of other computational systems can be compared

Turing machine fundamentals

- A tape divided into adjacent cells
  - each cell is inscribed with symbol from a finite alphabet
  - alphabet contains blank symbol and at least one other symbol
  - tape is arbitrarily extendible to the left and to the right
  - cells that have not been written to before are assumed to be filled with the empty symbol.
- A head to read & write symbols on the tape and move left & right
- A finite number of states
  - TM must be in exactly one state at any point in computation
  - computation begins in start state and ceases in halt state
- An action table tells TM what symbol to write, how to move the head and what its new state will be given the symbol it has just read on the tape and the state it is currently in
  - if there is no entry in the table for the current combination of symbol and state then the machine will halt

A Turing machine

R/W head moves left and right along the tape
The number displayed represents its current state, which can change as it computes

0 0 0 1 1 1 0 1 0 1

The tape is an infinite sequence of cells
Each cell contains a symbol (possibly blank)

Turing computable

- Is this really enough to compute everything?
- TM is theoretical device
  - infinite amount of memory
  - computation time is irrelevant
  - can represent data however we choose, provided the set of symbols is finite
- All attempts to characterize computation have turned out to be equivalent
- Functions that can be computed by a Turing machine are said to be Turing computable

Church-Turing Thesis

Every effective computation can be carried out by a Turing machine

Turing-computable ⇔ computable

- A function is computable if it can be computed by a Turing machine
  - or, a function is computable of a program exists that can compute it
  - or, a function is computable if there is an effective algorithm for evaluating the function which, given any input values for which the function is defined, produces the correct result and halts in a finite amount of time
- Q: Is there anything a Turing machine cannot do?
- A: Yes!
Noncomputability

- A function is **noncomputable** if there exists no Turing machine that could compute it.
- or, a noncomputable function is any function that cannot be computed by any program.
- many more functions then there are programs
- not possible to have a program for every function

A noncomputable problem

- Problem: List all subroutines that input an integer and return an integer
  - here is one short subroutine:
    ```plaintext
    int sub1(int x)
    { return 1; }
    ```
  - and here is another:
    ```plaintext
    int sub2(int x)
    { return 2; }
    ```
  - Can make infinitely many such subroutines

Halting

- Programs that run forever do not halt
  - programA
  - programs with infinite loops
- Programs may halt on some inputs but not on others
  - programB
- Programs with no loops always halt
  - programs composed of assignment statements, if statements and the return statement

Halt or not?

- Will the following code eventually terminate?
  ```plaintext
  while(x > 1)
  { 
    if( x % 2 == 0) 
      x = x / 2;
    else 
      x = 3*x + 1;
  }
  ```
  - What if x is 8? How about 7? How about any number > 0?

The halting problem

- Q: Is it possible to write a Java program to answer the following question
  - Does a given program halt on all inputs?
- It would be useful if we could write a program or compiler to test if a given source program contains an infinite loop
- If we had such a program, then we would never execute a program with an infinite loop
- This problem is known as the halting problem

![Halting Problem Diagram](https://via.placeholder.com/150)

- The input program is tested for halting.
- If it halts, the program halts.
- If it does not halt, the program does not halt.
- The process repeats until all possible inputs are tested.
- If the program halts on all inputs, it is said to be **halt**.
- If it does not halt on all inputs, it is said to be **no**.
The halting problem is noncomputable

No finite program can be written that will check other programs and halt in a finite time giving a solution to the halting problem

- There is no single finite program that will answer the halting question for all programs

Proof by contradiction

- How did Turing prove that such a program is theoretically impossible?
- Proof by contradiction:
  - Assume there is an algorithm \texttt{DoesHalt}(P, I) that returns "yes" if program \( P \) halts when given input \( I \) and "no" otherwise
  - Show that this assumption leads to a contradictory situation
  - Conclude that our initial assumption – the existence of a program to tell if another program halts – must be false
- Great Ideas, pages 438-440

Programs that read programs

Program \[ \begin{array}{c} \text{input} \\ \text{Program} \\ \text{output} \end{array} \]

- Almost every problem related to the behavior of programs is noncomputable
  - programs to check for property \( X \) in the behavior of all other programs
  - halting, equivalence, printing, correctness
- We can write programs to check almost any syntactic feature of programs
  - a compiler is a program that reads programs
  - we can measure the length of programs, number of statements, characters, arithmetic expressions

Computable or noncomputable?

1. Write a program that inputs a sequence of characters and returns true if it is a legal Java program
2. Write a program that inputs a sequence of characters and returns true if any permutation of these characters is a legal Java program
3. Write a program that inputs a program in any programming language and returns true if the characters can be rearranged to form a legal Java program
4. Write a program that inputs a Java program and returns true if the program computes the sum of its inputs
5. Write a program that inputs two Java programs and returns true if both programs always produce the same output

Computable or noncomputable?

1. Computable
2. Computable
3. Computable
4. Noncomputable
5. Noncomputable

Not impossible, but impractical

- Towers of Hanoi
  - how long to move \( n \) disks?
- What combination of switches turns the light on?
  - try all combinations, how many are there?
Program execution time

- How long a program takes to perform computations on large inputs
- Computable problems are basically divided into two classes
- Computations that can be completed in a reasonable time period are called **tractable**
  - a computation is tractable if its running time on N inputs is logarithmic or polynomial in N
- **Intractable** computations are those that cannot be realistically completed except for small examples
  - a computation is intractable if its running time on N inputs is exponential in N

Examples

- Tractable problems
  - searching or sorting N inputs
  - computing the sum of positive integers ≤ N
  - finding the largest element in a N×N array
  - counting the number of characters in a Java program
- Intractable problems
  - computing all permutations of N inputs
  - Towers of Hanoi
  - Traveling salesperson
  - Bin packing
  - Graph coloring

Tractable or intractable?

1. Write a program that inputs a sequence of characters and returns true if it is a legal Java program
   - **tractable**
2. Write a program that inputs a sequence of characters and returns true if any permutation of these characters is a legal Java program
   - **intractable**
3. Write a program that inputs a program in any programming language and returns true if the characters can be rearranged to form a legal Java program
   - **intractable**

Traveling salesperson

- A traveling salesperson must visit each of some number of cities before returning home
  - knows the distance between each of the cities
  - wants to minimize the total distance traveled while visiting all of the cities
- In what order should the salesperson visit the cities?

Bin packing

- Pack a given set of items having different sizes into a minimum number of equal-sized bins

Graph coloring

- Color the vertices of a graph such that no edge connects two identically colored vertices using the minimum number of colors

Example:
- Pack trucks with barrels using the smallest possible number of trucks
- Ideas?
Complexity classifications

- Class $\text{P}$
  - consists of all decision problems that can be solved on a deterministic sequential machine in time polynomial in the size of the input

- Class $\text{NP}$
  - non-deterministic, polynomial time
  - consists of all decision problems whose positive solutions can be verified in polynomial time, or
  - solution can be found in polynomial time on a non-deterministic machine

- Biggest open question in computer science: Is $\text{P} = \text{NP}$?

Are hard problems easy?

- Easy problems are in $\text{P}$, hard problems are in $\text{NP}$
- $\text{P} = \text{NP}$? Rich or famous?
- If the answer is yes, the whole class of hard problems can be solved efficiently
  - TSP and bin packing are “equally hard”
  - one problem is reducible to the other
- If no, none of the hard problems can be solved efficiently
- Widely conjectured that the answer is no
- $\text{P versus NP}$: CMI Millennium Prize Problem
  - first person to solve will be awarded $1,000,000$
  - http://www.claymath.org/millennium/P_vs_NP