Introduction to NP Completeness

Chapter 9

CPTR 318

Polynomial-time Algorithms

• Most of the algorithms we have seen so far have been polynomial-time algorithms
• Input size \( n \) => worst-case running time of \( n^k \), where \( k \) is a constant

Can all problems be solved in polynomial time?

Some Problems Cannot be Solved at All

• Write a computer program (or procedure or algorithm) \textit{Halt} that accepts two inputs:
  – \( p \), the string representation of a computer program that accepts a single string as input
  – \( i \), a string that serves as input to \( p \)

\begin{verbatim}
bool Halt(String p, String i)
{
    bool result = false;
    if ( isValidProgram(p) ) // The magic goes here!
        if ( program p halts when run on input string i )
            result = true;
    return result;
}
\end{verbatim}

Halt Parameters

• \( p \) is just a string of symbols:
  – String of characters for source code, or
  – String of bytes for compiled machine code
• Both of these representations are ultimately bitstrings, we can say that \( p \) can be boiled down to a single integer
• \( i \) is also ultimately a bitstring that maps to a single integer

\begin{verbatim}
bool Halt(String p, String i)
{
    bool result = false;
    if ( isValidProgram(p) ) // The magic goes here!
        if ( program p halts when run on input string i )
            result = true;
    return result;
}
\end{verbatim}

What does Halt Do?

• Halt determines if \( p \) terminates when given input \( i \)
• Said another way, Halt determines if \( i \) causes \( p \) to go into an infinite loop

\begin{verbatim}
bool Halt(String p, String i)
{
    bool result = false;
    if ( isValidProgram(p) ) // The magic goes here!
        if ( program p halts when run on input string i )
            result = true;
    return result;
}
\end{verbatim}

Pictorially

\begin{verbatim}
bool Halt(String p, String i)
{
    bool result = false;
    if ( isValidProgram(p) ) // The magic goes here!
        if ( program p halts when run on input string i )
            result = true;
    return result;
}
\end{verbatim}
The Halting Problem

- Alan Turing, 1936

Undecidability

- The Halting Problem is undecidable
  – that is, no algorithm can solve it

Perhaps it just a hard problem, and no one has yet to think up a solution

A Computer Program as Input?

- Is this even possible?
- Yes, consider
  – compilers
  – code formatters

So, what’s the problem?

Program Loop_If_Halts

Proof

- We must prove that no such algorithm exists
- The proof by contradiction:
  – Suppose such a algorithm, Halt, exists
  – Devise a new algorithm, Loop_If_Halts:

```cpp
bool Loop_If_Halts(String p) {
    if ( Halt(p, p) )
        while ( true )
          {} // Loop forever
    return true;
}
```

• If p corresponds to a valid program that accepts a single bitstring as an argument, then Halt is used to see if p halts on itself
• If p halts on p, then Loop_If_Halts never returns (it goes into an infinite loop)
• If p does not halt on p, then Loop_If_Halts terminates and returns true

Proof

Program Loop_If_Halts

```cpp
bool Loop_If_Halts(String p) {
    if ( Halt(p, p) )
        while ( true )
          {} // Loop forever
    return true;
}
```
A Clever Application of Loop\_If\_Halts

- What about calling Loop\_If\_Halts(Loop\_If\_Halts)?

<table>
<thead>
<tr>
<th>Loop_If_Halts</th>
<th>Halt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop_If_Halts</td>
<td>false</td>
</tr>
<tr>
<td>Loop_If_Halts</td>
<td>true</td>
</tr>
</tbody>
</table>

- That is, run Loop\_If\_Halts on itself
- Pass to the executing program the bitstring representing its own encoding

The Contradiction

- If Loop\_If\_Halts(Loop\_If\_Halts) does not terminate, then it terminates
- If Loop\_If\_Halts(Loop\_If\_Halts) terminates, then it does not terminate

Ramifications of the Halting Problem

- You may be able to construct a routine to address the halting problem that works in limited situations, but...
- You cannot devise a Halt procedure that will work under all circumstances
- Given any proposed Halt procedure you can devise an input that will cause it to fail
- Many interesting questions about computer programs are equivalent to the halting problem:
  - Will a particular section of code be executed?
  - Will a program halt on all input?
  - Will a program halt on any input?

Tractable vs. Intractable Problems

- While some problems are impossible, some are just hard
- Problems with algorithmic solutions that run in polynomial time are considered tractable
  - tractable = easy
  - efficient solutions
- Problems with algorithmic solutions that run in superpolynomial time are considered intractable
  - intractable = hard
  - Inefficient solutions

Tractable vs. Intractable Problems

- Shortest path vs. longest path
  - Shortest path: Dijkstra’s Algorithm $O(|E| \cdot |V|)$
  - Longest path (even if all edge weights are 1; that is, the edges are unweighted) intractable?
- Euler tour vs. Hamiltonian cycle
  - Euler tour: a cycle that traverses each edge in a graph exactly once (vertices may be revisited) $O(|E|)$
  - Hamiltonian cycle: a simple cycle that contains every vertex in the graph intractable?
P vs. NP

- P is the set of problems that are solvable in polynomial time
  - \( O(n^k) \) algorithms, where \( k \) is a constant, are known for all such problems
- NP is the set of problems whose solution can be checked in polynomial time (polynomial in the size of the input to the original problem)
  - A certificate of the solution is used for the check
- P is a subset of NP

Solution Certificate

- In the Hamiltonian cycle problem, the certificate would be the sequence \( H = <v_1, v_2, v_3, \ldots, v_V> \)
- Note that in polynomial time one can verify that
  - The length of \( H \) is \(|V|\)
  - All elements of \( G \) appear in \( H \)
  - For all \( i = 1, 2, 3, \ldots, |V| - 1 \)
    - \((v_i, v_{i+1})\) is in \( E \)
    - \((v_{|V|}, v_1)\) is in \( E \)

NP-complete Problems

- Some NP problems are special
- An NP-complete problem is as hard as any other problem in NP
- If a polynomial solution can be found for an NP-complete problem, all NP problems can be solved in polynomial time
  - Any problem in NP can be converted into an instance of an NP-complete problem in polynomial time

NP-complete Problems

- No polynomial-time algorithms have been found to solve any NP-complete problem
- No one has proven that a polynomial-time algorithm does not exist for an NP-complete problem
- The search has been going on for over 40 years!
- Superficially, several NP-complete problems appear very similar to problems that have polynomial-time solutions
- The big question: P ≠ NP?

Is NP-completeness a Problem?

- NP-complete problems crop up more often than you might think (Remember many appear to be very similar to "easy" problems)
- If you can show that a problem in NP-complete, you know that it is quite possibly intractable
- If it is NP-complete, do not waste your time trying to devise an exact solution to the general problem (A lot of smart people have been trying unsuccessfully for over 40 years!)
- The problem must be solved—what are your options?
  - Devise an approximation algorithm (Not the exact solution but good enough)
  - Concentrate on a tractable special case
How to Show a Problem is NPC

1. Take a known NP-complete problem (there are hundreds)
2. Demonstrate a process that converts an instance of the NP-complete problem into an instance of your problem in polynomial time

That proves your problem is NP-complete

Why?