CS 341: ALGORITHMS
Lecture 3: reductions
Readings: see website
Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

PROBLEM REDUCTIONS
Why reinvent the wheel? Reduce to another problem that you have already solved.

2SUM PROBLEM
• Input: Array \(A = [A[1], \ldots, A[n]]\) of integers and a target \(T\)
• Output: true if there exist two values in \(A\) (possibly the same value twice) whose sum equals \(T\), else false

Since the output is true/false, this is called a "decision problem"

Additional definitions:
- A yes-instance is an input to a decision problem, for which the correct output is true
- A no-instance is an input to a decision problem, for which the correct output is false

SIMPLE (BRUTE FORCE) SOLUTION
Runtime \(\Theta(n^2)\) by similar arguments to earlier...

Idea: let’s turn the innermost loop into something more efficient...

AN IMPROVEMENT
Instead of looping over \(j\), search the array for \(T - A[i]\)

How to do this efficiently?

IMPROVED ALGORITHM
Use binary search:

- Searches \(n\) elements in \(O(\log n)\) time
- Requires elements to be sorted!

VS. linear search, which takes \(O(n)\) time

What is this algorithm’s time complexity?
TIME COMPLEXITY
- Loop: iterations * work per iteration
  - Θ(n) + Θ(log n) = Θ(n log n)
- Entire algorithm: Θ(n log n) + Θ(n log n) = Θ(n log n)

PREPROCESSING
- The sort is an example of pre-processing
- It modifies the input to permit a more efficient algorithm (binary search as opposed to linear search)
- Note that a pre-processing step is only done once

3SUM PROBLEM
- Output: true if there exist three values in A (possibly taking the same value two or three times) whose sum equals T, else false

This is quite similar to 2SUM...
Can we reduce to 2SUM?

REDUCTIONS
- Suppose we already have a solution to 2SUM called Solve2SUM
- Suppose we design an algorithm Reduce3SUMto2SUM that solves 3SUM, and this algorithm calls Solve2SUM as a subroutine
- Reduce3SUMto2SUM is called a reduction from 3SUM to 2SUM
- Could also process input / call Solve2SUM multiple times
- If 3SUM can be reduced to 2SUM, we denote this by 3SUM \leq 2SUM

Mnemonic: 2SUM goes into 3SUM as a subproblem

REDUCTION FROM 3SUM TO 2SUM
- How can we use Solve2SUM to solve 3SUM?
- By changing the array A somehow?
- By changing the target T somehow?

```java
1. Reduce3SUMto2SUM(A[1..n], T)
2. for i = 1 .. n
3. T2 = T - A[i]
4. if Solve2SUM(A, T2) return true
5. return false
```

```
T = 9  A  1  2  3  2  1  3  2
i = 1  T2 = 8  Solve2SUM(A, 8) → False
i = 2  T2 = 16  Solve2SUM(A, 16) → False
i = 3  T2 = 11  Solve2SUM(A, 11) → False
i = 4  T2 = 9  Solve2SUM(A, 9) → False
i = 5  T2 = 7  Solve2SUM(A, 7) → False
i = 6  T2 = 10  Solve2SUM(A, 10) → False
i = 7  T2 = 6  Solve2SUM(A, 6) → True
```
REDUCTION CORRECTNESS
- Must prove: 3SUM(A, T) ⇔ ∃i : 2SUM(A, T - A[i])
- In other words,
  - Let A,T be any input to 3SUM
  - There exist a[i], A[j], A[k] that sum to T if and only if
  - there exists some A[m] such that Solve2SUM(A, T - A[m]) returns true

REDUCTION RUNTIME
1. Reduce3SUM to 2SUM (A[1...n], T)
2. for i = 1 ... n
3. T2 = T - A[i]
4. if Solve2SUM(A, T2) return true
5. return false

θ(n) loop iterations
Each iteration does Θ(1) + Runtime(Solve2SUM) work
Runtime depends on implementation of Solve2SUM!
- Brute force: Θ(n^3) = Θ(n^2 log n)
- Binary search: Θ(n) + Θ(n log n) = Θ(n^2 log n)

FAST 2SUM
T = 23 A
1 2 3
^ ^ 
→ → 
 sum = 24 too large! move j
sum = 22 too small! move i
sum = 23 just right!
*Correctness*
- Invariant: if there exists a solution i' < j' then i' ≥ i and j' ≤ j
- Exercise: fill in the proof details

FURTHER IMPROVEMENT
Recall our fastest Solve2SUM took \(O(n \log n)\) time for sorting, and \(O(n \log n)\) total time for searching
Can actually improve 2SUM to \(O(n)\) searching time with a greedy approach
Does not change complexity of 2SUM, but we will see we can still speed up our 3SUM reduction...

FAST 3SUM TO 2SUM REDUCTION
Although fast 2SUM is still \(O(n \log n)\), we can sort only once in our reduction
1. Reduce3SUM to 2SUM (A[1...n], T)
2. sort(A)
3. for i = 1 ... n
4. T2 = T - A[i]
5. if Fast2SUM(A, T2) return true
6. return false
Since 2SUM is given a pre-sorted array, it takes \(O(n)\) time!
- We get runtime \(O(n \log n) + O(n)O(n) = O(n^2)\)
IS THERE A FASTER 3SUM ALGORITHM?

- For many years, people thought this was likely optimal
- However faster algorithms appeared in 2014, 2017

Best known solution is:

\( O(n \log n)^{1+\epsilon} \)

This is a polylog factor faster than \( O(n^2) \)

... we suspect there is no solution faster than \( O(n^{2-\Omega(1)}) \)

PROGRESSIVELY HARDER WORKED EXAMPLES

A TRIVIAL REDUCTION

Suppose we want to multiply two integers, \( x \) and \( y \)

Consider the algebraic identity:

\[ xy = \frac{(x+y)^2 - x^2 - y^2}{2} \]

This allows us to show that \( \text{Multiplication} \leq \text{Squaring} \)

Oracle: ComputeSquare
- Oracle "gives" you a solution to the subproblem...
- If you solve ComputeSquare, you've solved Multiply

A MEDIUM REDUCTION

3SUMZero problem
- Input: array \( A = [A[1], \ldots, A[n]] \) of integers

Suppose we have solved 3SUMZero and want to solve 3SUM

It is straightforward to modify any algorithm for 3SUMZero so it solves 3SUM

Another approach is to find a reduction 3SUM \leq 3SUMZero. This would allow code re-use.

3SUM \leq 3SUMZero

- If and only if \( 3A[i] + 3A[j] + 3A[k] - 3T = 0 \)
- If and only if \( 3A[i] - T + (3A[j] - T) + (3A[k] - T) = 0 \)
- This suggests the following approach

\[
\begin{align*}
&\text{Reduce} \_\_ \text{SUM} \_\_ \text{SUMZero}(A, T) \\
&\text{for } i = 1 \ldots n \\
&\text{R[i] = A[i] - T} \\
&\text{return Solve3SUMZero(R)}
\end{align*}
\]

A HARD REDUCTION

3array3SUMZero problem
- Input: three arrays of \( n \) integers: \( A, B \) and \( C \)
- Output: true if there exist \( A[i], B[j], C[k] \), whose sum equals 0, else false

Let's try to reduce this to 3SUMZero

\[
\begin{align*}
&\text{TryReduce3arraySUMZeroTo3SUMZero}(A, B, C) \\
&\text{A' = concatenation of } A, B, C \\
&\text{return Solve3SUMZero(A')}
\end{align*}
\]

Problem: Solve3SUMZero might choose \( \geq 2 \) elements from the same array!
The Reduction

To show that this reduction is correct, we prove:
- **Case 1:** Assume true is the correct output for Solve3SUMZero(A') if and only if true is the correct output for Solve3array(A, B, C).
  - Want to show true is the correct output for Solve3array(A, B, C).
  - By our assumption, there exist A[i] + B[j] + C[k] = 0.
  - So, 10A[i] + 10B[j] + 10C[k] - 3 = 0.
  - So true is the correct output for Solve3SUMZero(A').

Correctness of the Reduction (1/3)

To show that this reduction is correct, we prove:
- true is the correct output for Solve3SUMZero(A') if and only if true is the correct output for Solve3array(A, B, C).

Correctness of the Reduction (2/3)

- Case 2: Assume true is the correct output for Solve3SUMZero(A')
  - Want to show true is the correct output for Solve3array(A, B, C).
  - By our assumption, there exist A[i] + B[j] + C[k] = 0.
  - Claim: this sum consists of one element from each of A, B, and C.
  - By cases... Example case: suppose, for contradiction, that A[i], A[j], A[k] are all in B.
  - Consider the sum modulo 10... only way to get it is to pick one element from each of A, B, C.
  - So, there is a sum i, with one from each of A, B, C.
  - So true is the correct output for Solve3array(A, B, C).

Correctness of the Reduction (3/3)

So, 3array3SUMZero ≤ 3SUMZero.
MANY-ONE REDUCTIONS

The previous three reductions had a very special structure
- We transformed (reduced) an instance of the first problem
to an instance of the second problem
- We called the oracle once, on the transformed instance

Reductions of this form, in the context of decision problems,
called many-one reductions
- (also known as polynomial transformations or Karp reductions)

We will many examples of these in the section on intractability