CS 341: ALGORITHMS

Lecture 3: reductions
Readings: see website

Trevor Brown (co-taught with Anna Lubiw)
https://www.student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca
Why reinvent the wheel?
Reduce to another problem that you have already solved.
2SUM PROBLEM

• Input: Array $A = [A[1], ..., A[n]]$ 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

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**

Since the output is **true/false**, this is called a “decision problem”
**SIMPLE (BRUTE FORCE) SOLUTION**

```python
def 2SUM_BruteForce(A[1..n], T):
    for i = 1 .. n
        for j = i .. n
                return true
    return false
```

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!

```
1 2SUM_Improved(A[1..n], T)
2     sort(A)
3     for i = 1 .. n
4       j = binary search for T-A[i]
5       in the subarray A[1..n]
6       if search is successful return true
7     return false
```
TIME COMPLEXITY

\[ \Theta(n \log n) \]

\[ \Theta(\log(n)) \]

\[ \Theta(n) \text{ iterations} \]

\begin{algorithm}
\begin{algorithmic}
\State 2SUM_Improved(A[1..n], T)
\State \hspace{1cm} sort(A)
\State \hspace{1cm} for i = 1 .. n
\State \hspace{1.5cm} j = \text{binary search for } T-A[i]
\State \hspace{1.5cm} in the subarray A[1..n]
\State \hspace{1.5cm} if search is successful return true
\State return false
\end{algorithmic}
\end{algorithm}

- **Loop:** iterations * work per iteration
  - \( \Theta(n) \times \Theta(\log n) = \Theta(n \log n) \)

- **Entire algorithm:** \( \Theta(n \log n) + \Theta(n \log n) = \Theta(n \log n) \)
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

• **Input:** Array $A = [A[1], ..., A[n]]$ of integers and a target $T$

• **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.

```plaintext
1 Reduce3SUMto2SUM(input_to_3sum)
2   input_to_2sum = process(input_to_3sum)
3   return Solve2SUM(input_to_2sum)
```

• `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?

```plaintext
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
```
Reduce3SUMto2SUM(A[1..n], T)
for i = 1 .. n
    T2 = T - A[i]
    if Solve2SUM(A, T2) return true
return false

\[
\begin{array}{c|c|c|c|c|c|c|}
 T  &=& 9 & A & 1 & -7 & -3 & 0 & 2 & -1 & 3 & -2 \\
 i  &=& 1 & \quad & T2 = 8 & \quad & \text{Solve2SUM}(A, 8) \rightarrow \text{False} \\
 i  &=& 2 & \quad & T2 = 16 & \quad & \text{Solve2SUM}(A, 16) \rightarrow \text{False} \\
 i  &=& 3 & \quad & T2 = 11 & \quad & \text{Solve2SUM}(A, 11) \rightarrow \text{False} \\
 i  &=& 4 & \quad & T2 = 9 & \quad & \text{Solve2SUM}(A, 9) \rightarrow \text{False} \\
 i  &=& 5 & \quad & T2 = 7 & \quad & \text{Solve2SUM}(A, 7) \rightarrow \text{False} \\
 i  &=& 6 & \quad & T2 = 10 & \quad & \text{Solve2SUM}(A, 10) \rightarrow \text{False} \\
 i  &=& 7 & \quad & T2 = 6 & \quad & \text{Solve2SUM}(A, 6) \rightarrow \text{True} \\
\end{array}
\]
REDUCTION CORRECTNESS

• **Must prove:** $3\text{SUM}(A, T) \iff \exists i : 2\text{SUM}(A, T - A[i])$

• In other words,

• Let $A,T$ be any input to $3\text{SUM}$

• There exist $A[i], A[j], A[k]$ that sum to $T$
  **if and only if**

• there exists some $A[m]$ such that $\text{Solve2SUM}(A, T - A[m])$ returns true
REDUCTION CORRECTNESS 2

- **WTP:** \( \exists A[i], A[j], A[k] \) that sum to \( T \) if and only if
- \( \exists A[m] \) such that \( \text{Solve2SUM}(A, T - A[m]) \) returns true

```plaintext
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
```

  - \( \iff \exists i : \text{Solve2SUM}(A, T - A[i]) \) returns true
REDUCTION RUNTIME

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`

- $\Theta(n)$ loop iterations
- Each iteration does $\Theta(1) + \text{Runtime}(\text{Solve2SUM})$ work
- Runtime depends on implementation of Solve2SUM!
- Brute force: $\Theta(n) \times \Theta(n^2) = \Theta(n^3)$
- Binary search: $\Theta(n) \times \Theta(n \log n) = \Theta(n^2 \log n)$
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 2SUM**

\[ T = 23 \quad A \begin{bmatrix} 2 & 3 & 5 & 11 & 12 & 20 & 22 \end{bmatrix} \]

- **Correctness**
- Invariant: if there exists a solution \( i' < j' \) then \( i' \geq i \) and \( j' \leq j \)
- Exercise: fill in the proof details

\begin{itemize}
  \item sum = 24
  \item too large! move \( j \)
  \item sum = 22
  \item too small! move \( i \)
  \item sum = 23
  \item Just right!
\end{itemize}

However, if \( i \) and \( j \) meet, there is no solution
FAST 3SUM TO 2SUM REDUCTION

• Although fast 2SUM is still $\Theta(n \log n)$, we can sort only once in our reduction

```plaintext
1  Reduce3SUMto2SUM(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 $\Theta(n)$ time!

• We get runtime $\Theta(n \log n) + \Theta(n)\Theta(n) = \Theta(n^2)$
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:
  \[ O(n^2 (\log \log n)^{O(1)} / \log^2 n) \]
- 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:
  \[
  xy = \frac{(x+y)^2 - (x-y)^2}{4}
  \]

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

\begin{verbatim}
1  ReduceMultiplyToSquare(x, y)
2  s = ComputeSquare(x+y)
3  t = ComputeSquare(x-y)
4  return ((s-t)>>2)
\end{verbatim}

• \textbf{Oracle}: \texttt{ComputeSquare}
  
  • Oracle “gives” you a solution to the \texttt{subproblem}…
  
  • If \texttt{you} solve \texttt{ComputeSquare}, you’ve solved \texttt{Multiply}
A MEDIUM REDUCTION

• **3SUMZero** problem
  • Input: array \( A = [A[1], ..., A[n]] \) 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 ≤ 3SUMZER0

- 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

```
1 Reduce_3SUM_to_3SUMZero(A, T)
2   for i = 1..n
3     B[i] = 3*A[i] - T
4   return Solve3SUMZero(B)
```

Given an oracle that solves 3SUMZero, let’s solve 3SUM

Can we find \( \sum x_i = T \) by finding \( \sum y_i = 0 \)?

Use the oracle to solve subproblem 3SUMZero(B)
If this reduction is correct, the result should be a solution to problem 3SUM(A, T)
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

```python
TRY_reduce_3array3SUMZero_to_3SUMZero(A, B, C)
A' = concatenation fo A, B, C
return Solve3SUMZero(A')
```

Problem: Solve3SUMZero might choose ≥ 2 elements from the same array!

Is this reduction correct?
No correct way to get a zero sum!

1. TRY_reduce_3array3SUMZero_to_3SUMZero(A, B, C)
2. A' = concatenation fo A, B, C
3. return Solve3SUMZero(A')

But Solve3SUMZero(A') returns true!

How to prevent picking 2+ elements from a subarray?
Somehow ensure the sum cannot be zero unless we pick one element from each subarray

Multiply by 10; Preserves sets of elements that sum to 0

Add +1
Add +2
Add -3

If sum of 3 elements is 0, so is their sum mod 10
So, only way to get 0 is to pick one from each subarray!
THE REDUCTION

`Reduce_3array3SUMZero_to_3SUMZero(A, B, C)`

```plaintext
for i = 1..n
    D[i] = 10A[i] + 1
    E[i] = 10B[i] + 2
    F[i] = 10C[i] - 3
A' = concatenation of D, E, F
return Solve3SUMZero(A')
```
CORRECTNESS OF THE REDUCTION (1/3)

```plaintext
Reduce_3array3SUMZero_to_3SUMZero(A, B, C)
    for i = 1..n
        D[i] = 10A[i] + 1
        E[i] = 10B[i] + 2
        F[i] = 10C[i] - 3
    A' = concatenation of D, E, F
    return Solve3SUMZero(A')
```

• 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)

• To show that this reduction is correct, we prove:
  true is the correct output for \texttt{Solve3SUMZero}(A') \textbf{if and only if}
  true is the correct output for \texttt{Solve3array}(A, B, C)

• Case 1: Assume true is the correct output for \texttt{Solve3array}(A, B, C)
  • Want to show true is the correct output for \texttt{Solve3SUMZero}(A')
  • By our assumption, there exist \( A[i] + B[j] + C[k] = 0 \)
  • So \( 10A[i] + 1 + 10B[j] + 2 + 10C[k] - 3 = 0 \)

  \begin{align*}
  & D[i] + E[j] + F[k] \\
  \end{align*}

  • So true is the correct output for \texttt{Solve3SUMZero}(A')
CORRECTNESS OF THE REDUCTION (3/3)

- **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 $A'[i] + A'[j] + A'[k] = 0$
  - 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 $B$

Then the sum $A'[i] + A'[j] + A'[k] = 10B[..] + 2 + 10B[..] + 2 + 10B[..] + 2$, which is **not zero**! Contradiction!

Consider the sum modulo 10... only way to get 0 is to pick one element from each of $A, B, C$

So, there is a sum=0, with one from each of $A, B$ and $C$. So, true is the correct output for Solve3array($A, B, C$)

So, $\text{3array3SUMZero} \leq \text{3SUMZero}$
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, are called many-one reductions
  • (also known as polynomial transformations or Karp reductions)
• We will many examples of these in the section on intractability