CS 341: ALGORITHMS

Trevor Brown
trevor.brown@uwaterloo.ca
THIS TIME

- (Turing) reductions
  - 3SUM problem and variants as examples
REDUCTIONS
3SUM PROBLEM

• Input: Array $A = [A[1], ..., A[n]]$ of distinct integers
• Output: true if there exist three distinct values in $A$ whose sum equals 0, else false
• Can anyone suggest a simple solution?

Since the output is true/false, this is called a “decision problem”
What is the running time of this algorithm?

3 loops, each doing ~n steps $\rightarrow O(n^3)$ steps

Other terms for running time:
- time complexity
- step complexity
AN IMPROVEMENT

• Loop over all possible i and j as before
• Instead of looping over k, search the array for -(A[i]+A[j])

How to do this efficiently?
IMPROVED ALGORITHM

- Use binary search:
  - Searches $n$ elements in $O(\log n)$ time
  - Requires elements are sorted!

```c
3SUM_Improved(int A[1..n])
   sort(A)
   for i = 1 .. n-2
      for j = i+1 .. n-1
         k = binary search for -(A[i]+A[j]) in the subarray A[j+1..n]
         if search is successful return true
   return false
```

Compare with linear search, which takes $O(n)$ time

What is this algorithm’s time complexity?
TIME COMPLEXITY

- **Inner loop**: iterations * work per iteration
  - \( O(n) \times O(\log n) = O(n \log n) \)
- **Outer loop**: \( O(n) \times \) inner loop
  - \( O(n) \times O(n \log n) = O(n^2 \log n) \)
- **Entire algorithm**: \( O(n \log n) + O(n^2 \log n) = O(n^2 \log n) \)
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
FURTHER IMPROVEMENT

- Even better way to use the sorted array
- We start with $j = i+1$ and $k = n$
- At any stage of the algorithm, we either increment $j$ or decrement $k$
Consider the sorted array $-11 \ -10 \ -7 \ -3 \ 2 \ 4 \ 8 \ 10$. 

Proof of optimality posted by Doug!
**EVEN FASTER PSEUDOCODE**

```
3SUM_Fast(int A[1..n])
    sort(A)
    for i = 1 .. n-2
        j = i+1
        k = n
        while j < k
            if sum < 0 then j = j+1
            else if sum > 0 then k = k-1
            else return true
    return false
```

- Time complexity: sort + outer loop * inner loop
  - \( O(n \log n) + O(n) \times O(n) = O(n \log n) + O(n^2) = O(n^2) \)

Time complexity?

Stop if \( j \) and \( k \) meet. They get closer in each iteration. \( O(n) \) iterations!

O(1) steps in each iteration
REDUCTIONS

• Suppose $P$ and $Q$ are problems, and $A_Q$ is a hypothetical algorithm that solves $Q$
  • $A_Q$ is called an oracle; it is a black box that gives answers

• Suppose you design an algorithm $R$ to solve $P$, and this algorithm calls $A_Q$ as a subroutine
  • Then $R$ is called a reduction from $P$ to $Q$
    • It reduces the problem of solving $P$ to the problem of solving $Q$. (Why?)
      • If we have a solution to $Q$, then $R$ gives us a solution to $P$
      • If $P$ can be reduced to $Q$, we denote this by $P \leq Q$

a “real” implementation of the oracle $A_Q$
A SIMPLE 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 Multiplication ≤ Squaring

```
1. ReduceMultiplyToSquare(int x, int y)
2.     s = ComputeSquare(x+y)
3.     t = ComputeSquare(x-y)
4.     return \((s-t)/4\)
```

• Oracle: ComputeSquare
  • If you solve ComputeSquare, you solve Multiply

Note that “division by 4” here is just bit-shifting by 2.
A MORE ADVANCED REDUCTION

• **Target3SUM** problem
  • Input: an array $A$ of $n$ distinct integers, and a **target integer** $T$
  • Output: true if there are three distinct elements in $A$ whose sum equals $T$, else false

• It is straightforward to modify any algorithm for **3SUM** so it solves **Target3SUM**

• Another approach is to find a reduction $\text{Target3SUM} \leq \text{3SUM}$. This would allow code re-use.
TARGET3SUM \leq 3SUM

- if and only if \( (A[i] - T/3) + (A[j] - T/3) + (A[k] - T/3) = 0 \) (as long as \( T \) is divisible by 3)
- This suggests the following approach

```python
1  Reduce_Target3SUM_to_3SUM(A, T)
   for i = 1..n
2  return Solve3SUM(B)
```

Modification: the transformation \( B[i] = 3A[i] - T \) works for any integer \( T \)
COMPLEXITY ANALYSIS

Suppose we implement the oracle Solve3SUM with one of our “real” 3SUM algorithms.

What is the complexity of the resulting algorithm?

- If we plug in 3SUM_BruteForce, we get $O(n + n^3) = O(n^3)$
- If we plug in 3SUM_Improved, $O(n + n^2 \log n) = O(n^2 \log n)$
- If we plug in 3SUM_Fast, $O(n + n^2) = O(n^2)$
MANY-ONE REDUCTIONS

• The previous reduction 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
A MORE ADVANCED REDUCTION

• **3array3SUM** problem
  • Input: three arrays of n distinct 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 3SUM

```python
1 TRY_Reduce_3array3SUM_to_3SUM(A, B, C)
2    A' = concatenation of A, B and C
3    return 3SUM(A')
```

Is this reduction correct?

Problem: 3SUM might choose $\geq 2$ elements from the same array!

How can we ensure that 3SUM picks one element form each subarray?
A BETTER ATTEMPT

• Input: three arrays of n distinct integers: A, B and C
• Output: true if there exist A[i], B[j], C[k], whose sum equals 0, else false

```
Reduce_3array3SUM_to_3SUM(A, B, C)
  for i = 1..n-1
    D[i] = 10A[i] + 1
    E[i] = 10B[i] + 2
    F[i] = 10C[i] - 3
  A' = concatenation of D, E and F
  return 3SUM(A')
```

How to argue 3SUM picks one element per sub-array?
CORRECTNESS OF THE REDUCTION (1/2)

• To show that this reduction is correct, we prove: 
  true is the correct output for 3SUM(A') if and only if 
  true is the correct output for 3array3SUM(A, B, C)

• Case 1: Assume true is the correct output for 3array3SUM(A, B, C)
  • Want to show true is the correct output for 3SUM(A')
  • By our assumption, there exist $A[i] + B[j] + C[k] = 0$
  • So $10A[i]+1 + 10B[j]+2 + 10C[k]-3 = 0$
  • So $D[i] + E[j] + F[k] = 0$ (since $D[i] = 10A[i]+1$, …)
  • So 3SUM(A') returns true
CORRECTNESS OF THE REDUCTION (2/2)

**Case 2:** Assume true is the correct output for 3SUM(A')

Want to show true is the correct output for 3array3SUM(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

Example case:
Suppose, for contradiction, that $A'[i], A'[j], A'[k]$ are elements of 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

```
1 Reduce_3array3SUM_to_3SUM(A, B, C)
2     for i = 1..n-1
3         D[i] = 10A[i] + 1
4         E[i] = 10B[i] + 2
5         F[i] = 10C[i] - 3
6     A' = concatenation of D, E and F
7     return 3SUM(A')
```
NEXT TIME

• Divide and conquer paradigm
• Merge sort
• Recurrence relations
  • Tree recursion method: merge sort as an example
  • Guess and check method