Homework 21

Brian Knotten, Brett Schreiber, Brian Falkenstein

October 17, 2018

6

3-Sum $\leq_{n \log n}$ 3-Sum-Part

3-SumAlgorithm($z_1, ..., z_n$)
    Let $A = \{z_1, ..., z_n\}$
    Let $B = \{z_1, ..., z_n\}$
    Let $C = \{z_1, ..., z_n\}$
    Return 3-Sum-PartAlgorithm(A, B, C)

Since 3-Sum-Part still contains the restriction that $i \neq j \neq k$, then the existence of a 3-Sum all from one set is equivalent to the existence of a 3-Sum of three equal sets. Since the transformation of the input takes $O(n)$ time, and since there is no transformation of the boolean output, the overall algorithm preserves the $O(n \log n)$ runtime.

3-Sum-Part $\leq_{n \log n}$ 3-Sum

3-Sum-PartAlgorithm(A, B, C)
    Let $A' = A \cdot 2 + 3$; that is, for each element in A we are multiplying by 2 and adding 3.
    Let $B' = B \cdot 2 + 5$; that is, for each element in B we are multiplying by 2 and adding 5.
    Let $C' = C \cdot 2 - 8$; that is, for each element in C we are multiplying by 2 and subtracting 8.
    Let S = $\{A', B', C'\} = \{a'_1, ..., a'_n, b'_1, ..., b'_n, c'_1, ..., c'_n\ \}$
    Return 3-SumAlgorithm(S)

This reduction works assuming all elements in A, B, C are integers. Because every element in A and B is being doubled and then increased by a positive amount, the sum of any three elements from A or any three elements from B is being increased by a positive amount, preventing the sum from being 0. By the same logic, decreasing every element of C decreases the overall sum of its elements, preventing it from being 0. Because every element is doubled (and is an integer), the edge case of the sum of the elements of one of the collections being 0 after transformation is prevented - ex: $A = \{-3, -3, -3\}$.
Since the transformation takes $O(n)$ time, and there is no transformation of the boolean output, the overall algorithm preserves the $O(n \log n)$ runtime.

3-Sum $\leq_{n \log n}$ 3-Colinear

This is a reduction we learned in class.

3-SumAlgorithm($z_1, ..., z_n$):
    for $i = 1$ to n do:
        Let $p_i = (z_i, z_i^3)$
    Return 3-ColinearAlgorithm($p_1, ..., p_n$)

As discussed in class,

$$\begin{aligned}
\frac{b^3-a^3}{b-a} &= \frac{c^3-b^3}{c-b} \\
\frac{(b-a)(b^2+ba+a^2)}{b-a} &= \frac{(c-b)(c^2+cb+b^2)}{c-b} \\
b^2+ba+a^2 &= c^2+cb+b^2 \\
b^2+ba+a^2-c^2-cb-b^2 &= 0 \\
ba+a^2-c^2-cb &= 0 \\
b(a-c)+(a-c)(a+c) &= 0 \\
(a-c)(a+b+c) &= 0 \\
a+b+c &= 0
\end{aligned}$$

3-Colinear-Line $\leq_{n \log n}$ 3-Colinear

An instance of 3-Colinear-Line can be thought of as a special case of an instance of 3-Colinear with an extra restriction of not being true when the only colinear line in the input has a slope of 0. So the algorithm is as follows:

3-Colinear-LineAlgorithm$(p_1, ..., p_n)$ Return 3-Colinear-Line$(p_1, ..., p_n)$ where horizontal lines have been filtered out.

3-Colinear-LineAlgorithm retains $O(n \log n)$ runtime because there is no transformation of input to output.

3-Colinear-Line $\leq_{n \log n}$ 3-Sum

If an instance of 3-Colinear-Line is true, then it must have three points $(a, 0)(b, 1), (c, 2)$, which are colinear, since this problem does not allow horizontal lines as solutions. Therefore the slope between a and b must be the same as the slope between b and c as follows:

$$\begin{aligned}
\frac{2-1}{c-b} &= \frac{1-0}{b-a} \\
\frac{1}{c-b} &= \frac{1}{b-a} \\
c-b &= b-a \\
a-2b+c &= 0
\end{aligned}$$

So there must exist three numbers a, $-2b$, and c in an instance of 3-Sum which makes it true. So try manipulating each possible b in the 3-Colinear-Line instance to get an output for 3-SumAlgorithm. So the algorithm for 3-Colinear-Line is as follows:

3-Colinear-LineAlgorithm$(p_1, ..., p_n)$:
For $i = 1$ to n do:
Let $(x_i, y_i) = p_i$ for $i = 1$ to n.
If 3-SumAlgorithm$(x_1, ... - 2x_i, ...x_n)$ then return True
Otherwise return false.

3-Sum-Part $\leq_{n \log n}$ 3-Colinear-Line

Since 3-Colinear-Line requires that solution points all have different y-coordinates, and since a solution to 3-Sum-Part requires that each term comes from a different set, it seems best to transform the input of 3-Sum-Part by turning each term from A into $(a_i, 0)$, each term from B into $(b_i, 1)$, and each term from c into $(c_i, 2)$. Then, the points are fed to the 3-Colinear-Line algorithm. If true, then output true. Otherwise, try again with A's points at 1 rather than 0, and B's points at 0 rather than 1. Continue permuting until the 3-Colinear-Line outputs true. If not, output false.

Summary

Starting with a poly-time algorithm for 3-Sum, we can obtain an algorithm for 3-Colinear-Line and 3-Sum-Part.
Starting with a poly-time algorithm for 3-Sum-Part, we can obtain an algorithm for 3-Sum, which in turn gives an algorithm for 3-Colinear-Line.
Starting with a poly-time algorithm for 3-Colinear, we can obtain an algorithm for 3-Colinear-Line and 3-Sum, which in turn gives an algorithm for 3-Sum-Part.
Strating with a poly-time algorithm for 3-Colinear-Line, we can obtain an algorithm for 3-Sum-Part, which in turn gives an algorithm for 3-Sum.

7

The only instances of LineSplit which output false are ones where every line segment intersects another line segment so that you could, for example, draw every line segment on the plane without lifting your pencil off the page. So in a reduction from 3-Sum, a 3-Sum instance does not contain 3 terms that sum to 0 if and only if the instance of LineSplit has the property described above. Then, the 3-Sum algorithm can return the result of the call to the LineSplit algorithm. Since the reduction must be done in $O(n \log n)$, that implies that the transformation should only require one pass over the 3-Sum input. This means that the transformation is local, such as every number in 3-Sum being transformed to a line segment independently. Beyond this, we could not find the transformation.