Meet-in-the-Middle

Matthew Savage
Adapted from a blog post on infoarena

October 10, 2014

1 Introduction

1.1 4-Sum Problem

You are given n ($4 \leq n \leq 5000$) integers. Determine whether or not exactly four of them can be summed to zero (though you may use the same element multiple times). For example, for this input:

```
2 6 1 3 -2 -7 -9
```

The answer is yes, because $6 + 3 + (-2) + (-7) = 0$. Alternatively, $3 + 3 + 3 + (-9) = 0$ is another valid solution.

1.2 Naive Solution

The naive solution to this problem is to try all sets of four numbers. While this will always get the correct answer, choosing four numbers will take $O(n^4)$ - far too slow for $n \leq 5000$.

1.3 Slight Optimization

Let's call the four numbers a, b, c, and d. Through some simple manipulation,

$$\begin{aligned} a + b + c + d &= 0 \\ a + b + c &= -d \end{aligned}$$

Let's use this information to our advantage. If we store all possible outcomes of adding three numbers ($a + b + c$), then we only need to do a single check for each of the values in the original set (because if $-d$ is in the set, then the four numbers add to 0). Since we're only choosing three numbers and then doing a constant-time search for each element (use a hash table), the complexity becomes $O(n^3 + n) = O(n^3)$ - better, but still not fast enough for the upper bound.

1.4 Meet-in-the-Middle

As the name implies, the *meet-in-the-middle* approach says that we should meet the ends of the problem halfway, in this case by storing the sums of pairs of numbers. That way we can check the n^2 choices for $(a+b)$ against the n^2 choices for $(c+d)$, for a complexity of $O(n^2)$. This is fast enough for the bounds.

2 A More Formal Statement

The meet-in-the-middle technique is a technique used to decrease the time complexity of an algorithm *exponentially* for a tradeoff increase in space complexity.

In the most formal sense, meet-in-the-middle is a technique to increase the speed of graph searches. Therefore, in order to use meet-in-the-middle, you need to have

1. a set of possible states;
2. a set of possible state transitions;
3. an initial state;
4. a final state; and
5. a maximum searching distance.

The meet-in-the-middle technique states that the time complexity of such a search can be improved by searching for half of the maximum distance from both the intial and final states instead of by searching for the full distance from only the initial state.

In the case of a straight graph, the first four of these are (1) the nodes, (2) the edges, (3) the start node, and (4) the target node.

3 Six Degrees of Separation

Imagine a graph of all users of Facebook. An edge exists between two people if they are friends, and we can define the distance between two users to be the minimum number of edges you must travel to get from one to the other. (Thus, friends have distance 1, friends of friends have distance 2, etc.)

What is the most time-efficient way of determining whether or not two target users are within distance 6 of each other?

The naive method (which is extremely inefficient) would be to use a straight BFS up to a depth of 6. If the actual distance between the two targets is k and the average *degree* (nodewise edge count) of the graph is p, the search will process $O(p^{max(k,6})$ nodes.

Using meet-in-the-middle, we can run two depth-3 searches. The first step is to run a search starting at one of the two targets up to a depth of 3, storing some kind of data (perhaps a reference in a hash table) for each visited node. This will take $O(p^3)$ time.

Next, we run another depth-3 search starting at the other target. For each visited node, check if the node is in the table stored during the first search. If it is, then there exists a path of at most length 3 from one target to this middle node, and another path of at most length 3 from this middle node to the other target, so there must exist a path of at most length 6 between the two target nodes. This second search also runs in $O(p^3)$ time, so the total runtime for the algorithm is $O(p^3)$.

4 The Problem with 2-DES

DES is an encryption method that uses 56-bit keys. By today's standards, that's considered fairly insecure - methods exist of cracking this through brute force relatively quickly.

An easy way of strengthening the encryption is to apply DES twice, with two different 56-bit keys. This provides for 112 bits of entropy instead of 56. This causes the time required to brute force it naively to increase by a factor of 2^56.

However, using meet-in-the-middle, we can go much faster. Consider what the graph looks like. The set of all possible states is in three groups: all possible plaintexts, all possible "middletexts" (after the first DES is applied but before the second), and all possible ciphertexts. The state transitions are applications of DES. The intial state a known plaintext, and the final state is a known ciphertext that matches that plaintext.

Using the same logic as with the 4-sum problem, we can compute all possible encryptions of the plaintext, store them, and then try matching them with all possible decryptions of the ciphertext. That is, rather then solving it this way:

$$\text{plaintext} \xrightarrow{Encrypt\ with\ K_1} \text{middletext} \xrightarrow{Encrypt\ with\ K_2} \text{ciphertext}$$

We're solving it this way:

$$\text{plaintext} \xrightarrow{Encrypt\ with\ K_1} \text{middletext} \xleftarrow{Decrypt\ with\ K_2} \text{ciphertext}$$

Since we only run each value once for each step of encryption, the number of encryption-decryption operations is reduced from 2^{112} to 2^{56}.

5 Example Problems

1. **String Reversals** - You are given two strings, S and T. Determine whether or not you can get to string T by starting at string S and applying at most four substring reversal operations.
2. **Sliding Puzzle** - Consider a sliding puzzle consisting of a 3 by 3 grid of squares with one slot missing. The other eight squares are labeled 1-8. The puzzle is solved when the pieces are rearranged such that the top row is 1-2-3, the middle row is 4-5-6, and the bottom row is 7-8-empty. Given a starting board, determine whether or not the puzzle is solvable, and if so output a solution. *(HINT: The puzzle is either unsolvable or is solvable in less than 31 moves from every possible starting position.)*
3. **Square Fence** - You are given n planks of (given) varying lengths that you must arrange into a square fence without overlapping or breaking any of them. How can you arrange the planks to accomplish this? *(HINT: the complexity will be $O(4^{n/2})$.)*