Contents

1 Some Bruteforce Optimizations

1.1 Constrained Bruteforce

Sometimes a seemingly impossible problem (due to timelimit) with higher time complexity can be solved by slightly changing a bruteforce solution. This comes down to observing some unusual constraints (often involving pigeonhole principle) in the problem that allows a tuned bruteforce to pass.

Problems:

- https://codeforces.com/contest/1501/problem/C
- https://codeforces.com/contest/1554/problem/B
- https://codeforces.com/contest/1553/problem/E

1.2 Meet In the Middle

Some solutions with exponential or worse time-complexity can be optimized by dividing into two parts and adjusting the solution for both parts and combining them.

Resources:

- http://www.shafaetsplanet.com/?p=1756

Problems

- https://lightoj.com/problem/coin-change-iv

1.3 Prefix Sum Optimization

Bruteforce solutions are often optimized by using dynamic programming or data structures. Prefix sum optimization of DP is an example of both working together. When the subproblems of a state involves a range of states, then the ranges can be calculated fast using a prefix sum DP.

Problems

- https://codeforces.com/gym/103185/problem/K
- https://codeforces.com/gym/102861/problem/A

1.4 Two Pointers

Some solutions can be optimized using a sliding window technique, usually an outer n^2 loop can be reduced to an outer n loop using two sliding pointers.

Problems

- https://codeforces.com/contest/1555/problem/E
- https://codeforces.com/contest/1537/problem/E2
- https://codeforces.com/contest/1494/problem/C

Harmonic Series

```
for(int i = 0; i < n; i++)
    for(int j = i; j < n; j+= i)
        // some O(1) task
```

What is the time complexity of this code? This can be expressed as

$$T(n) = \sum_{i=0}^{n} \frac{n}{i} \approx nlog(n) \tag{1}$$

This comes useful in some problems, often in divisor related problems. Seive of eratosthenes uses this fact as well.

DP/DFS on Trees:

Often DP on trees are simple DFS without memoisation due to the lack of overlapping subproblems. For unrooted trees, it is usually the case that you'd need to root it arbitrarily and when needed, build the table bottom up and consider the parent edge separately.

problems:

- https://codeforces.com/contest/1554/problem/E