Talk - Representing Things as Graphs

Alex Li

January 2020

1 Introduction

Graphs are cool, but aren't they a bit too random general math thing? In this talk, we will see how graphs can appear even in situations where you never expect to find them, and how finding graphs in these situations can lead to huge simiplifications of otherwise difficult problems.

2 Palindrome XOR

2.1 Statement

https://codeforces.com/problemset/problem/1147/D

Given a length m string s of the characters $0, 1, ?$ how many ways can we choose two base two palindromic integers $a < b$ such that $1 \leq a, b < 2 \oplus m$, and $a \oplus b$ match s? We say that the number $a \oplus b$ matches s if, for all k, the kth digit in base 2 of $a \oplus b$ is equal to the kth character in s, or the kth character of s is ?.

Since the answer can be big, output modulo 998244353. Goal: Do it in time quadratic in m.

2.1.1 Understanding the statement - XOR

The bitwise XOR of two numbers a and b, denoted $a \oplus b$, is the binary number such that, for all k, the kth digit is 0 iff the kth digit of a and the kth digit of b are the same.

Example. $11010 \oplus 1100 = 00110 = 110$.

2.1.2 Understanding the statement - Examples

If s is the string 10110, how many ways are there? Since $b > a$, we notice that $b \geq 10000$, $a < 10000$. By the palindromic property, $b = 1???1$, and then by the XOR property, $a = 0???1$. By the palindromic property, we know that a starts with a 1. If $a = 1??1$, then $b = 11??1$, so $b = 11?11$, so $a = 1001$, and then $b = 11111$.

We could also have $a = 1?1$, then xoring fixes $b = 100?1$, $b = 10001, a = 111$. We can have $a = 11$, and this fixes the value of b to be $m^a = 10101$. Finally we have $a = 1$, then $b = 10111$, but this is not a palindrome.

Notice how the palindrome and XOR property severly restrict the number of choices. Here, all of our freedom comes from us choosing the length of the string a. Next we consider an example with ? symbols, and see how our freedom is restricted.

Suppose $s = 1???$. Then we know $b = 1??1$. If the length of a is 1 or 2, then the 2 question marks can be anything so long as they are the same. If the length of a is 3 then we get 1?1, and we can choose any value for that question mark in addition to any value for the pair of question marks in b, so there are 4 different ways for a total of 8.

2.2 Solution Intuition

From the examples we see that, after fixing the length of a, we get a set of free variables, and then we can add to the answer $2^{\#free}$. However, how exactly can we count the number of free variables? To do this we need to somehow get a more program friendly definition of our constraints.

You know what programs like to eat? Graphs! We can represent this problem as a graph with every bit of a and b as verticies. Our constraints of XOR and palindrome basically say that two bits need to be the same or different. We can represent these constraints as edges beteween the nodes. Notice that we now have a very nice interpretation of 'the number of free variables': it is the number of connected components of the graph! Well, not quite - we need to add a few more constraints. In particular, bits of b that are not paired with bits of a might need to be set to be either 0 or 1 depending on the given string, and the first bit of a has to be 1. To represent these constraints, we can add an edge between all these nodes and a new node which represents a bit that is always 0.

Nice! Now, there is one problem - we count the number of free variables, but only if the probem is solvable. This layout would allow impossible answers, so we need to check that it's possible to satisfy the constraints. If it is possible, then we can choose any of the free bits and it will set the rest. To check for impossibility, we add weights to our graph. We mark the weight as 0 if two edges are the same and 1 if two edges are different. We want to check that our graph can be split up into 2 components. Edges between these components have weight 1, and edges from one component to itself has weight 0. This problem can be solved greedily by depth-first searching the graph- we are forced to mark a node by our decision on the last node. Note this is the same solution for checking if a graph is 2-colorable/bipartite.

2.3 Analysis

Overall we fix the length of a to be any of m numbers, then do a dfs on a graph that has $O(m)$ edges + nodes. So the total time is $O(m)$

2.4 Code

//https://codeforces.com/contest/1147/submission/69638106

3 Subset with 0 Sum

3.1 Problem

//https://codeforces.com/contest/1270/problem/G You are given n integers $a_1, a_2, \dots, a_n$, such that for each $1 \leq i \leq n$ holds $i - n \leq a_i \leq i - 1$

Find some nonempty subset of these integers, whose sum is equal to 0. It can be shown that such a subset exists under given constraints. If there are several possible subsets with zero-sum, you can find any of them.

3.2 Examples

Consider the sequence $[-1, -1, -1, 3]$. If we take the whole sequence, the sum is 0.

Consider the sequence $[-9, 1, -7, -6, 2, 2, 1, 1, 5, 1]$. We notice that -6 + 5 + 1 = 0.

3.3 Brainstorm

How can we solve this at all?

3.4 Graphy Solution

Build a directed graph with n verticies, corresponding to the indicies of the given values. Let the ith vertex point to the vertex $i - a - i$. Then any cycle is the solution to the problem. And a cycle will surely exist in this situation.

Whoa slow down! How do we know that a cycle is the solution? Suppose the cycle is $i_1, i_2, i_3, \dots i_m$. Then we have $i_1 - a_{i_1} = i_2, i_2 - a_{i_2} = i_3, \dots i_n - a_{i_m} = i_1$. Summing these equalities, we get $\sum_j^m i_j + a_{i_j} = \sum_j^m i_j$, so $\sum_j^m a_{i_j} = 0$.

So we have a $O(N)$ solution!

3.5 Heuristic solution

This problem is quite difficult, so you maybe can just try to do some random stuff: Compute the prefix sums of the array. If two are equal, then the subarray between them is a valid solution. There are about n^2 different possible prefix sum values, if we consider the 'birthday paradox', it is not so unlikely that we can come up with a solution. Especially if we repeat this process reordering the array a whole lot randomly.

3.6 Making it real

The problem with the aforementioned heuristic is that it does not work every time. Sure, it works most of the time, but the problem is that the range of values can be huge, and so it will not work every time. This would be a valid solution if only, if only the possible values of the prefix sum could be in some smaller range of size $n-1$. In that case, we could surely find two prefix sums that are equal, but that case will never come):

Unless we take action and make it come! Let's try order the elements such that the prefix sums are all in the interval $[0, n)$. Can we do this? Surely we can make at least the first one in this interval, we just need to pick the nth element of the sequence. Then if the new value is k, we need to choose a value within the interval $[-k, n-k)$. Surely the $n-k$th element of the sequence satisfies this. And so on. But wait! This strategy seems great on paper, but what if we get to a prefix sum with value k twice? Surely we can't choose the element at index $n-k$ twice... Ah, but the entire goal is to get to the same prefix sum twice, so there is no reason to panic. Nice!

4 Take Home Problem

//https://codeforces.com/problemset/problem/1272/E