Tug-of-War

Set up

It was written in **C++ 11**. Run by command

```
g++ -lm -lcrypt -O2 -std=c++11 -pipe -DONLINE_JUDGE
```

It has been accepted by Uva.

Algorithm Overview

- It is not able to enumerate all possible solution since the test cases are two large.
- According to the lecture note, we need to design a dynamic programming to store the valid ways to make sum-weight j. j = 1,2, …,W/2, where W is the total weight of all the persons.
- Binay mentioned in the email, we can use bitmask to solve this question.

  > *If the kth bit of bitmask DP[j] is on, it represents that k number weights are needed to make sum j. We need to use long long integer to store the mask.*

- In addition, DP[0] should be initialized as 1 since there should be only 1 way to allocate the person who has weight 0.

  > *Initially, DP[0] = 1LL to indicate that 0 number of weights need to make sum weight 0.*

- Build DP[i,j] function and see the details below.
    - Please note, in DP building process, “i” is outer loop index which is between 0 to the number of people, “j” is inner loop index which is counting down from total-weight to i-th weight.
- After that, your nested loop to find out the total weights in the left team and right team. Left team should be the minimum weight and right team should be the maximum weight. And inner loop only need to loop from 0 to middle-weight instead of entire weight.

> *The turned-on bits of bitmask of DP[j] indicate the number of weights needed to make sum-weight j. We then look for the ceil(n/2) bit of the bitmask of DP[j] for j = W/2,(W/2) − 1, ...*

Data Structure and Data Type

- The recuirsive definition of DP[j], when i-th person is being considered.
  `DP[j] = DP[j] v (DP[j - weight[i]] << 1LL)`
- `vector<int>` to store all persons’ weight.
- `long long` as DP array type.

Constrained Circular permutation

Set up

It was written in C++. It was accepted by ACM-ICPC.

Algorithm Overview

Fix 1 on top of the clock. Then fix element on the left of 1 and element on the right of 1. Now we only consider the rest n-3 elements. DFS can be appiled recursivly to count the total valid permutation.

Detail

- Let’s mark the top 3 element c[1], c[n], c[2] as ABC
    - We have A = 1, A < B < C
    - Then when we have A < C < B repeated permutations. So we can ignore this case.
- DFS only need to increment total count when we hit the last element because we are counting the total permutation not number of triplets in a permutation.

Carmichael

Set up

It was written in C++ and it was accepted by UVa.

Algorithm Overview

- The number indicated in the problem is too big to calculate directly.

- We need to use the algorithm indicated in slide #14 of Number Theory to recursively calculate the exponent for Femat test.
- Other than that, we need to use a correct primarity test to determine whether the number is prime or not. We've used looping from 2 to `sqrt(N)` in this question.
- Once we have the result, we can determine whether it is a Carmichael number because Carmichael number is not prime and passes all the Femat tests.
- We can combine the results of the previous 2 points to make a decision.

Robot On Ice

Set up

It was written in C++ and it was accepted by UVa.

Algorithm Overview

- It is easy to fine a naive aproach for this problem.
    - Just start from the coordinate (0, 0) and see at each point, whether we can go left, right, top or bottom and create a recursive call of we can do so.
    - This way we've been generating all the paths that starts from (0, 0).
    - And then we can filter out paths that end with (0, 1) and goes through the 3 checkpoints calculated.

- However, this naive approach is too slow to work with a grid plane with 8 rows and 8 columns.
- In order to accelarate the search, we need to do pruning.
    1. We can check if we are at the 1/4, 1/2 and 3/4 steps of the execution. If indeed we are, and we are not at the checkpoints, that means the path we are walking is wrong and we stop the search.
    2. We can also check whether we are currently at the checkpoints. If we are, and we are not at the desired steps, stop the search of the current path because it is not a valid path as well.
    3. Another check would be to calculate Manhattan distance between the current point we are at and one of the checkpoint. If the distance is too large, meaning that the point is taking larger than the desired steps to go to, then we can stop the search.
    4. If a move results in disconnection of the grid plane, we can also stop the search because disconnection means some of the points will never be visited if we are going the current path. This is not acceptable as we need to visit every tile according to the question.
- After lots of such pruning, we can use the original algorithm to generate paths and keep the count number updated.
- Report the count number after the search.