Carmichael 10006

This question is about to test the number either is Carmichael number or not.
If the number is the Carmichael then this number is not a prime number and it can fit into this formula: a^n mod n =a; for any a smaller than n.

In the code, first I test the prime part, if the number is prime then set the flag to true and jump to the next input.

```
if(isPrime(input))
{
    isNormal=true;
}
else
{
    for(int i=2;i<input;i++)
    {
        if(modfunction(i,input,input)!=i)
        {
            //System.out.println(modfunction(i,input,input)+"   i is "+i);
            isNormal=true;
            break;
        }
    }
}
if(isNormal)
```

Then the next part is to test the mod part via a loop for all possible a, which I used a quick mod function to reduce the running time.

```
static int modfunction(long a, int n,int mod)
{
    long result = 1;
    while(n>0)
    {
        if((n&1)==1)//bitwise and
        {
            result=result*a%mod;
        }
            a=a*a%mod;


        n>>=1;//bit manipulation is equal to divide by 2 to lower, but the number of loops is upper.
    }
```

And return the result.

Tug of War 10032

Algorithm:
Dynamic Programming

bool dp[i][j][k]: if we only consider 1 to ith person, is it possible to make total weight of left side is j, right side is k

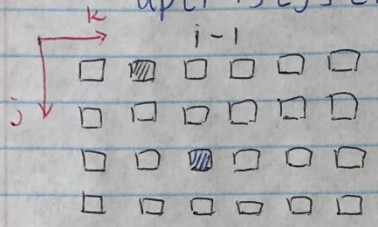
bool dp[i][j][k] = dp[i-1][j-w[i]][k-1] or dp[i-1][j][k]

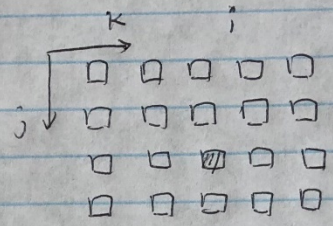
x = w/2 to 1, the first x such that
N is even:
    dp[N][x][N/2] is true
N is odd:
    dp[N][x][N/2] or dp[N][x][N/2 + 1] is true
However, 3-d array is way too complicated, so that 3-d array is compressed to 1-d array as shown in the graph.

bool dp[i][j][k] =

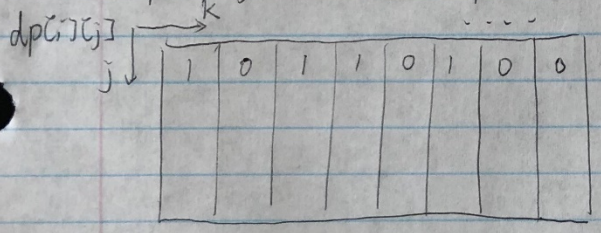
dp[i-1][j-w[i]][k-1] or

dp[i-1][j][k]



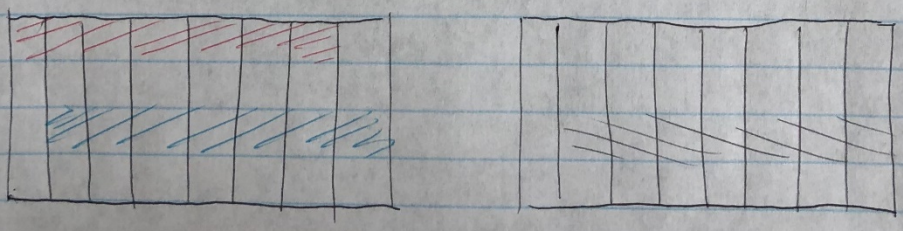


dp[i][j][k+0] = dp[i-1][j-w[i]][k-1] or dp[i-1][j][k+0]
dp[i][j][k+1] = dp[i-1][j-w[i]][k+0] or dp[i-1][j][k+1]
dp[i][j][k+2] = dp[i-1][j-w[i]][k+1] or dp[i-1][j][k+2]



ull dp[i][j] = 0bool 0|1 0|

ull dp[i][j] = (dp[i-1][j-w[i]] << 1) | (dp[i-1][j])



ull dp[j] = (dp[i-1][j-w[i]] << 1) | (dp[j])

Robot on Ice 1098

Data structure:

Queue, LinkedList, Point from awt

Algorithm:

1. Initial 2-D Arrays graph for record matrix and 3 target points with their cost (s1...s3), and start searching the path from (0,0)

2. Use breath first search keep search and valid the neighbor point, with global value count for calculate paths

3. there is 3 specific help method for the main bfs method solve(): isValid(), isDisconnect() and distance()

4. isValid() for make sure next step's row & col number are still in the range by grid size. isDisconnect is by using queue to valid the point is visited or not visit yet. The distance method is implement Pythagorean theorem for compare and update the shortest path/point.

5. keep update and return count value for each case

Constrained Circular Permutations 2608