DYNAMIC PROGRAMMING

PREREQUISITES:

1) Recursion
2) Complexity

"Those who cannot remember the past are condemned to repeat it"

- Dynamic Programming

INTUITION:

Quoting directly from Jonathan Paulson's answer(who is a Software Engineer at Google) to "How should I explain dynamic programming to a 4-year-old?" on Quora:

Writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper.
"What's that equal to?"
Counting "Eight!"
Writes down another "1+" on the left.
"What about that?"
"Nine!" " How'd you know it was nine so fast?"
"You just added one more!"
"So you didn't need to recount because you remembered there were eight! Dynamic Programming is just a fancy way to say remembering stuff to save time later!"

Let us take a look at this problem solving technique in more detail.

THE IDEA OF SUBPROBLEM OF A PROBLEM:

Can you write a recursive code to compute the factorial of a number?

What is the base case?
fact(0)=1

What is the recursive case?
fact(N) = fact(N-1) * N

You should be able to write the following code.
Link: https://ideone.com/Mh3nFC

If you observe this expression:
fact(N) = fact(N-1) * N ,
you will find that we need the value of fact(N-1) to compute fact(N).

So, we say that fact(N-1) is a sub problem of fact(N).
To calculate fact(N-1) we need the value of fact(N-2).
To calculate fact(N-2), we need the value of fact(N-3).
And so on….
So, fact(N-1), fact(N-2)..... fact(1) are all subproblems of fact(N).
In other words, to know the answer of fact(N) we need to know the answers of fact(N-1), fact(N-2),......fact(1).
Great!

OVERLAPPING SUBPROBLEMS:
Let us consider another problem.
Suppose I ask you to find the Nth fibonacci number recursively.

What is the base case? It is the case for which I already know the answer:
fib(1) = 1
fib(2) = 1

What is the recursive case?
fib(N)=fib(N-1) + fib(N-2)

What is the subproblem to compute fib(N)?
fib(N-1) and fib(N-2), right?

What is needed to compute fib(N-1)?
fib(N-2) and fib(N-3).

So fib(N-1), fib(N-2), fib(N-3).... fib(1) are the subproblems of fib(N).
We have to calculate these to compute fib(N).
You should be able to write the following code:
Link: https://ideone.com/LcNjpR
Let us draw a recursion tree to calculate fib(5):

```
                   fib(5)
                /          \
            fib(4)            fib(3)
          /     \            /    \
       fib(3)    fib(2)     fib(2)   fib(1)
      /    \      /   \     /   \
 fib(2)   fib(1)  fib(1) fib(0) fib(1) fib(0)
 /   \
fib(1) fib(0)
```

If you observe the recursion tree carefully, you will see that we are calculating the sub problems: fib(3) twice, fib(2) thrice, fib(1) 4 times etc.

How do we avoid this wasteful recomputation?

The moment we compute fib(3) or fib(2) what we do is store(remember) it in an integer array? This is called memoization. Next time when we want the value of fib(3) or fib(2), we don't need to recompute all over again.
We can simply read from memory.

Have a look at the following visualization:
https://www.cs.usfca.edu/~galles/visualization/DPFib.html
Instructions to visualize:

1. Type a number between 0 and 20 in the text field in the top left corner.
2. Click on Fibonacci Recursive. This is the visualization for the inefficient code you wrote.
3. Click on Fibonacci Memoized. This is the visualization which incorporates the idea of memoization to avoid wasteful recomputation.

THEN, WHAT IS DYNAMIC PROGRAMMING?

This is exactly what dynamic programming is. It is recursion + memoization.

The core idea of Dynamic Programming is to avoid repeated work by remembering partial results!

Is this all to learn about DP?
Then, why are people so hyped about it?
Is it is so simple?
Just write recursive solution and then memoize it and profit!
Yes, it is this simple!

What does state of a dp mean?
It is a way to describe a situation, a subsolution for the problem by using one or more parameters of any data type.
In our fibonacci example what is the dp state?
It is fib(i) , the ith fibonacci number.
Examples will make it more clear. In some problems it is the state of a dp which is hard to come up with. In some problems it won't be as simple as requiring only one integer parameter to describe the state. Many more parameters can be required of other data types.

A PROBLEM:

Consider the following problem:

Given a list of N coins, their values (V1,V2,, … , VN), and the total sum S. Find the minimum number of coins the sum of which is S (we can use as many coins of one type as we want), or report that it's not possible to select coins in such a way that they sum up to S.

Think for a minute how can you approach this problem.

We will write a dynamic programming solution (recursion with memoization) for the following problem.
Let the state of our dp be dp[i]=minimum number of coins required to represent sum i.

Base Case:
What is the base case? How do we decide it?
For what value(s) of S do we already know the answer(i.e the minimum number of coins the sum of which is S).

What if S is V1 , V2, …… VN? How many coins are required?
1, right?

So our base case is if S is one of the values of the N coins, then we can represent that amount using the coin of that value i.e
dp[V1] = 1
dp[V2] = 1
dp[V3] = 1
.
.
.
.
dp[VN] = 1
We definitely cannot do better than 1 coin!

Recursive Case:
Let us move on to the recursive case.
What is the minimum number of coins required to represent S?

Let us use a coin with value V1. In that case, what is the minimum number of coins required to represent S? 1+minimum number of coins required to represent (S-V1). And we recursively compute this i.e minimum number of coins required to represent (S-V1).

Let us use a coin with value V2. In that case, what is the minimum number of coins required to represent S? 1+minimum number of coins required to represent (S-V2).

Let us use 1 coin with value V3. In that case what is the minimum number of coins required to represent S? 1+minimum number of coins required to represent (S-V3).

.

.

.

Let us use 1 coin with value VN. In that case what is the minimum number of coins required to represent S? 1+minimum number of coins required to represent (S-VN).

So the minimum number of coins required to represent S is
1+min(dp[S-V1],dp[S-V2],dp[S-V3],………, dp[S-VN])

Let us look at how to convert all our ideas into code once we know the state of the dp, recursive and base cases.
Link: https://ideone.com/xyApFf

Reference:
You may refer to this link *Introduction(Beginner)*:
https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/ for a more detailed analysis of the same problem.

Visualisation:
Have a look at this excellent visualization here:
https://www.cs.usfca.edu/~galles/visualization/DPChange.html
Instructions:

1. Enter a value of S in the text field on the top left corner of screen.
2. Click Change memoized.
3. Then enter the same value of S in the text field on the top left corner of screen.
4. Click Change recursive
5. Which one is faster!? ;)

ANOTHER PROBLEM:
You still haven’t got the flavour of what amazing tasks can be solved efficiently using dynamic programming. Let us look at a more interesting problem.

Have a look at the first answer starting from subhead “Motivation Problem”, here:
https://www.quora.com/Are-there-any-good-resources-or-tutorials-for-dynamic-programming-besides-the-TopCoder-tutorial

This will give you an good idea as to what kind of problems can be solved using dynamic programming!

<u>CLASSIC DP PROBLEM #1:</u>

Let us see another classic dynamic programming problem:

Given a sequence of N numbers – A[0] , A[1] , …, A[N-1] . Find the length of the longest non-decreasing sequence.

Let us first decide the state of the dp i.e what parameters we need, to describe the situation of our problem.

<u>State:</u>
Let's the state be dp[i] = the longest non-decreasing sequence ending with number A[i] .

<u>Base Case:</u>
What is our base case? What is the answer that we already know?
Don't we know dp[0]?
What is dp[0]?
dp[0] is the longest non decreasing sequence ending at the 0th index. It is always 1!

<u>Recursive Case:</u>
Let us move onto the recursive case.
Let us consider an element say A[3].
Now we need to compute the longest non decreasing sequence ending at index 3.
How do we do it?
It is the maximum of the following cases:
1+dp[0], iff A[0]<=A[3]
1+dp[1], iff A[1]<=A[3]
1+dp[2], iff A[2]<=A[3]
^------>These conditions must be satisfied to get a nondecreasing sequence.

What we basically do is we consider all the elements before A[3] which is less than equal to A[3] and see the longest non decreasing sequence ending at that element. Then we add 1 to that answer to include A[3] in the sequence as well.

<u>Final Answer:</u>
Our longest non decreasing sequence of the given array will have it's ending point at one of the indices from 0 to N-1.
Therefore, the answer will be,
max(dp[0], dp[1], …...dp[N-1])

<u>Reference:</u>

Have a look at this tutorial *elementary section* for a more detailed solution:
https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/

CLASSIC DP PROBLEM #2:

Let us look at another classical problem.
But now I will take a different approach. I won't simplify the problem and solution from the original document. I will just give you a link and you have to figure it out on your own by reading the contents in that link.
Have a look at "Maximum Sum Subsection" in this tutorial.
http://www.iarcs.org.in/inoi/online-study-material/topics/dp-classics.php

Conclusion:
Remember one thing dynamic programming is a topic that requires immense practice. Solve as many problems as possible. Here we discussed only a few of the many variations possible. Whenever you are solving a dynamic programming problem try to come up with the dp state to describe the situation of the problem. Then figure out the base case, then the recursive case. Memoize and done!

Here are a few problems you can try:

1) https://www.codechef.com/problems/WORLDCUP
2) https://www.codechef.com/ZCOPRAC/problems/ZCO14002
3) https://www.codechef.com/ZCOPRAC/problems/ZCO14004

Make sure to try hard! If you are still unable to solve these, do give the editorials a read.

Further Reading:
http://www.iarcs.org.in/inoi/online-study-material/topics/dp.php
https://www.hackerearth.com/practice/notes/dynamic-programming-problems-involving-grids/

More resources:
https://www.commonlounge.com/discussion/52ae98d1a9e447f9a33968985e036a2a/main
https://www.commonlounge.com/discussion/3e462ec698ab4aaeb72af15bc8acb50f/main