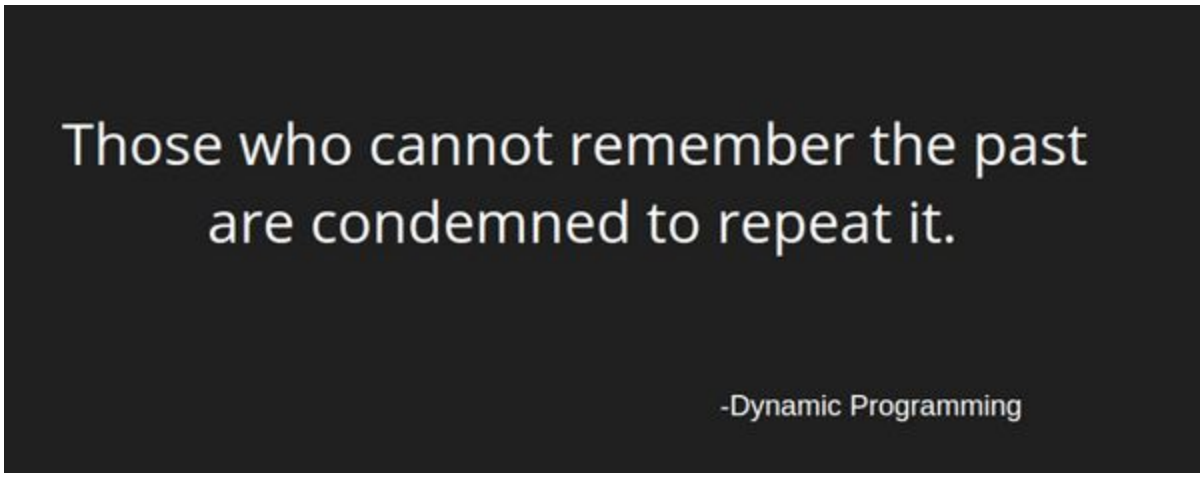


So, most of us have heard about this term “DP” a lot and have seen people in fear as soon as they listen to it.

I used to fear DP a lot as well, but I found a way to tackle it, since then I have started to fall in love with DP.

I’ll share my way :-)

I’ll make you fall in love with this topic.

Important points:-

1. To be good at DP, all you have to do is solve classical DP-problems and their variations.
2. Recognize patterns and improve upon your thinking skills in DP
3. Love DP, it will love you back !

There are many variations of DP and I'll discuss them all one by one so you can become a pro!

Introduction to DP:-

To find solution to a problem, we divide the problem into sub-problems, find answers to those sub-problems,combine them to get the original answer!

That's it!

Example:- Say I ask you to calculate :- (1+2+3+4+5)

You do this:-

1. Break it into sub-problems : (1+2) + (3+4) +(5)
2. Find answers to those sub-problems: (3) + (7) + (5)
3. Combine them to get the answer to the original problem : 15 :-)

That's what we call dynamic programming ! :-)

My personal trick :-

dp[i] usually mean the best answer to the problem till the i'th index of the array.

Obviously, final answer will be dp[n](where 'n' is the size of the array)

We cannot calculate dp[n] directly, we first need to calculate dp[1],dp[2],... and combine their results to find the value of dp[n] :-)

Problem-1 : We are given an array of integers(a[n]) . We are given multiple queries of the form : (1, i) which means we need to output the sum of all numbers from index- '1' to index 'i' of the array.

Analysis : Running a loop for each query and finding the sum is a good idea but not very efficient as it takes O(N*Q) time.

Lets create a dp-array of size 'n'.

dp[1]=sum of all numbers from (1,1)

dp[2]=sum of all numbers from (1,2)...

and so on.

Say, a[5]={4,5,3,2,1}...(assume 1-based-indexing here)

So, dp[1]=4(pretty easy).....(1)

dp[2]=4+5=9.....(2)

dp[3]=4+5+3=12.....(3) and so on.

So, for any index 'i' ,

dp[i]=a[i]+dp[i-1],

Example:-

dp[3] = a[3] + dp[3−1] = a[3] + dp[2]= 3 + 9 =12 ....(which is same as equation...(3))

Piece of code(C++) :-

```
dp[0]=0;

int i=1;

while(i≤n)

{

dp[i]=a[i]+dp[i-1];

i++;

}
```

This took O(N) time!

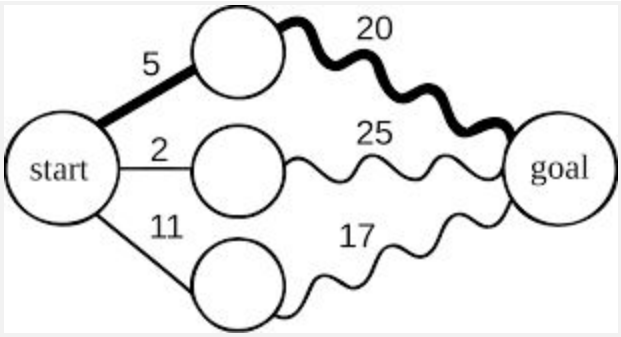
Voila! We did it!

So, now for all the queries, all we gotta do is output the value of dp[i], which we have already pre-computed :-)

Time taken:- O(N+Q) [O(N) for pre-computation, as we answered each query in O(1), total time taken was O(Q)]

So you get the basic idea of DP is to make a recurrence relation,and then run a loop,and calculate(pre-compute) the values :-)

My aim is to start this series from a very beginner level and make you reach the advanced level :-)



Problem-2:- Understanding this problem and its solution properly will make a strong foundation for you in the DP world .(This worked for me :-))

Here we go- Given an array of integers(positive as well as negative) ,select some elements from this array(select a subset) such that:-

1. Sum of those elements is maximum(Sum of the subset is maximum) .
2. No 2 elements in the subset should be consecutive.

Example :- {2,4,6,7,8}

Answer:- {2+6+8=16}

Common Trick : We create a dp-array , and dp[i] means the maximum sum we could get till index-'i' of the array.

For the above example,

dp[1] = 2 (2), [This is the best answer you could get if size of the array was one]

dp[2]= 4(4),[This is the best answer you could get if size of the array was two]

dp[3]=8(6+2),[This is the best answer you could get if size of the array was three].....lets call this equation-(1)...

dp[4]=11(7+4),[This is the best answer you could get if size of the array was four]

dp[5]=16(2+6+8),[[This is the best answer you could get if size of the array was five]

Now , the next thing that I am going to say, you gotta feel it deeply . If you are able to do it, you win!!

Say, I have calculated dp[1],dp[2] and dp[3] by pure observation.

Now, I have to calculate dp[4].

So I have only 2 choices:-

i)Include the 4'th element , if I do this, I can’t include the 3rd element as consecutive elements are not allowed, so ,

dp[4]= a[4]+dp[2].....(answer will be 4th element plus the best answer till index ‘2’ of the array)

ii)Exclude the 4th element, we don’t select it! So the previous answer is the current answer, we don’t change anything,

dp[4]=dp[3],

Final answer is the maximum of two choices :->

dp[4]=maximum(a[4]+dp[2],dp[3])

=maximum(7+4,8)

=maximum(11,8)

=11.....(verify this with equation...(1))

Voila, we did it !!!

So,the recurrence relation for this problem is,

for any ‘i’ ,

dp[i]=max(a[i]+dp[i-2],dp[i-1])

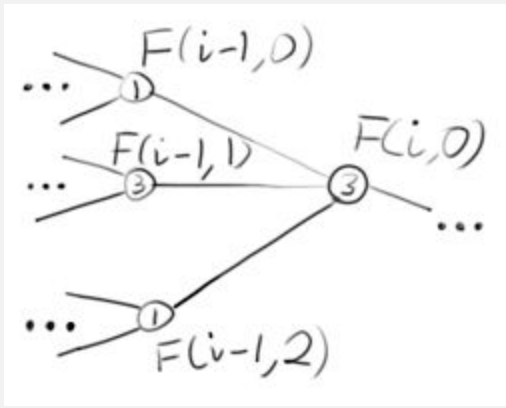
We will calculate dp[i] for all ‘i’ from (1 to n) using the above formula.

And dp[n] will be the maximum possible sum for the whole array!!

Woohoo!!!!

Some code:-

```
dp[0] = 0;

dp[1] = max(a[1],dp[0]);

i = 2;

while(i≤n)

{

dp[i] = max(dp[i-2]+a[i], dp[i-1]);

i++;

}
```



Aim of this series:- “ To make you fall in love with Dynamic Programming. To show you the actual beauty of this topic.To go from beginner level to master level.”

Some of the common tricks we have learned as of now :

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-’i’ of the array.

2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...
3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].
4. dp[n] means the answer for the whole array.

Last time, we discussed this problem

:-

Given an array of integers(positive as well as negative) ,select some elements from this array(select a subset) such that:-

1. Sum of those elements is maximum(Sum of the subset is maximum) .
2. No 2 elements in the subset should be consecutive.

Example :- {2,4,6,7,8}

Answer:- {2+6+8=16}

Now, lets solve a harder variation of this problem,which will make our understanding strong.

Modified Version : We are given ‘2’ arrays . Select some elements from both of these arrays (select a subset) such that:-

1. Sum of those elements is maximum(Sum of the subset is maximum).
2. No 2 elements in the subset should be consecutive.(Note:-If you select, say 5'th element from Array-1, then you are not allowed to select 4'th element from either Array-1 or Array-2 :-))

Example:-

Array 1(a1) : {1,5,3,21234}

Array 2(a2) : {-4509,200,3,40}

Answer:- (200+21234=21434)

Common Trick : We create a dp-array , and dp[i] means the maximum sum we could get till index-'i' of the array.

For the above example,

dp[1] = 1(1) , [This is the best answer you could get if size of the array was one]

dp[2]= 200(200),[This is the best answer you could get if size of the array was two]

dp[3]=200(200), [This is the best answer you could get if size of the array was three].....lets call this equation-(1)...

dp[4]=21434(200+21234), [This is the best answer you could get if size of the array was four]

Say, I have calculated dp[1] , dp[2] and dp[3] by pure observation.

Now, I have to calculate dp[4].

So I have only 2 choices:-

i)Include the 4'th element (of either array-1 or array-2, if I do this, I can't include the 3rd element(from both the arrays) as consecutive elements are not allowed, so,

dp[4]= (max(a1[4],a2[4])) + dp[2]).....(answer will be 4th element plus the best answer till index '2' of the arrays)

ii)Exclude the 4th element, we don't select it! So the previous answer is the current answer, we don't change anything,

dp[4]=dp[3],

Final answer is the maximum of two choices :->

dp[4]=maximum(max(a1[4],a2[4])+dp[2],dp[3])

=maximum(max(40,21234)+200,200)

=maximum(21234+200,200)

=maximum(21434,200)

=21434.....(verify this with equation...(1))

Voila, we did it !!!

So,the recurrence relation for this problem is,

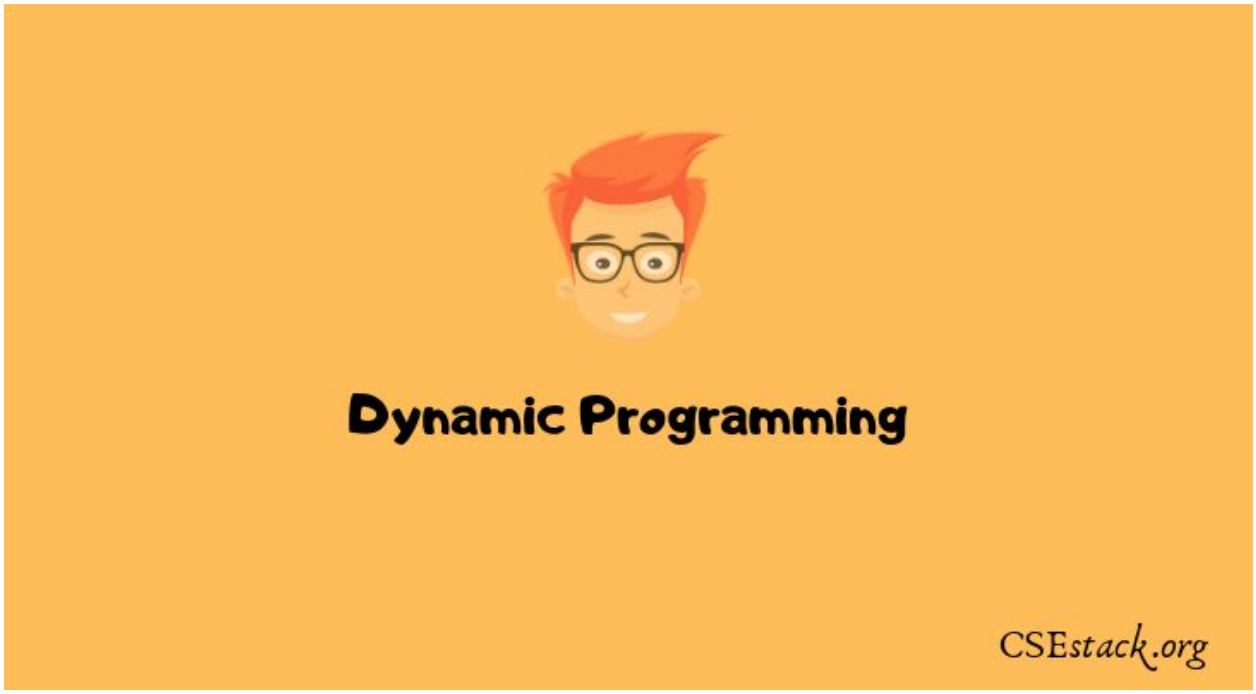
for any ‘i’ ,

dp[i]=max(max(a1[i],a2[i]) +dp[i-2],dp[i-1])

Some C++ code:-

```
//dp[1] and dp[2] is calculated by observation.

//we run a loop from i=3 to i=n , dp[n] is the final answer.

i=3;

while(i≤n)

{
```

```
dp[i]=max( max(a1[i],a2[i])+dp[i-2],dp[i-1]);

i++;

}

cout<<dp[n];
```



Aim of this tutorial : — " To make you fall in love with Dynamic Programming. To show you the actual beauty of this topic.To go from beginner level to master level,which will ultimately help you in competitive programming and coding interviews!"

What have we learned so far?

1)We always create a dp-array of size ’n’.

2) dp[i] is the best answer for the given problem till index ‘i’ of the array.

3)dp[n] is the final answer!!!

4)We write the values of dp[1],dp[2] by pure observation.(as its very easy to do!!)

5)We find a formula, example formula : dp[i]= 2*dp[i-1]+3*dp[i-2] , and apply this formula (run a for loop) to calculate dp[3],dp[4],….dp[n]…And then we are done!!

So let’s discuss a delicious problem today! We are given a number ‘x’ , we want to reduce it to ‘1’ in minimum number of steps possible!!

At each step, we are allowed to :

1)Decrement the given number by ‘1’……..(i)

2)Decrement the given number by ‘2’ ……(ii)

3)Divide the given number by ‘7’ only if it is divisible by ‘7’……..(iii)

So how do we solve this problem(find the minimum number of steps to reduce any given ‘x’ to ‘1’) ?

We chill and feel good about the precious life given to us by God, now we go back to the problem!

Check out the steps I taught above!!!

1)First we create a dp-array of size ‘x’.

2)dp[i] is the best answer for any number ‘i’.In other words, dp[i]=minimum number of steps required to convert the number ‘i’ to 1.

3)We will find dp[1],dp[2],dp[3],dp[4],……..dp[x].

4)dp[x] is the final answer!!!!

So, tell me, what is the value of dp[1]???

How many steps does it take to reduce '1' to '1'????

Answer : Zero.

Hence, dp[1]=0

Now, tell me, how many steps does it take to reduce '2' to '1'???

Answer: It's simple, 1 step is what it takes .(Just decrement 1 from '2'....refer(i) above!!)

Therefore,dp[2]=1.

Now, dp[3]?? 1st way : 3 — ->2 —
->1(Total-2-steps-taken!!)

2nd way : 3 — ->1(Only 1 step taken).

Hence dp[3]=minimum(1-step,2-steps)=1-step.

Now,dp[4]??

1st way : 4 →2 →1.(Decrement by 2, then decrement by 1)

2nd way : 4 →3 →2 — ->1(Decrement by "1" 3 times)

3rd way : 4 →3 — ->1(Decrement by 1, then decrement by 2)

Hence,dp[4]=2(2 steps).

Now, for dp[5], let's use some other trick!!

We already know that 5 can be reduced to '3' or '4' in the first step(DECREMENT EITHER BY '1' OR '2').

Say, we reduce it to "3". Now, we already know the best answer for "3" , which is dp[3]!!! Now ,the other choice is to reduce it to "4" in the first step itself. We already know the best answer for "4" , which is dp[4]!!! Eurekaa!!! Eureka!!! Eureka!!!

Hence, dp[5]=1+dp[3] ....(Why +1 ?? Because it took "1" step to convert five to three) OR dp[5]=1+dp[4] , Henceforth,

dp[5]=minimum(dp[4]+1,dp[3]+1)=

minimum(2+1,1+1)=minimum(3,2)=2!!!

So the formula is, dp[i]=minimum(dp[i-1]+1,dp[i-2]+1).

Lets now calculate , dp[14] for example .
dp[14]=minimum(dp[13]+1,dp[12]+1)..... BUT ARE we forgetting something???

Yes!!! “14” is divisible by “7” ,

so we can apply the third operation, as well, we may divide it by “7” 14 →14/7 — ->2 — ->1(Took only-2-steps!)

Hence, dp[14]= dp[14/7]+1 is one of the choices,

So the final formula for any number divisible by “7” is :
→ dp[i]=minimum(dp[i-1]+1,dp[i-2]+1,dp[i/7]+ 1)...

See, dp is so lovely, I love it more than my girlfriend ❤
You should too :P

Some pseudo code:-

dp[1]=0;

```
dp[2]=1;

dp[3]=1;

dp[4]=2;

i=5;

while(i≤x)

{

if(i%7==0)

{

dp[i]=min(dp[i/7]+1,dp[i-1]+1,dp[i-2]+1);

}
```

else

{

dp[i]=min(1+dp[i-1],1+dp[i-2]) ;

}

i++;

}

Related-problems:-

1)https://www.geeksforgeeks.org/reduce-a-number-to-1-by-performing-given-operations/

2)https://www.geeksforgeeks.org/minimum-number-of-operations-required-to-reduce-n-to-1/

3)https://www.geeksforgeeks.org/minimum-steps-minimize-n-per-given-condition/

The principal on which Dynamic Programming operates :- “We always find the most optimal aka the best solution using dynamic programming.”

Make sure you read all the previous Dp tutorials before you begin to binge learn this one.

Here are they:-

We solved this problem in Dp-Tutorial-2 :

Given an array of integers(positive as well as negative) , select some elements from this array(select a subset) such that:-

1. Sum of those elements is maximum(Sum of the subset is maximum) .
2. No 2 elements in the subset should be consecutive.

Example :- {2,4,6,7,8}

Answer:- {2+6+8=16}.

Let's change the second condition :-> Instead of 2, No 3 elements in the subset should be consecutive.

Now, what do we do ?

I taught you few tricks earlier and they are as follows:-

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-'i' of the array.

2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...

3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].

4. dp[n] means the answer for the whole array.

Example Array : {3000,2000,1000,3,11}

Answer:- {3000,2000,3,11}=sum()=5014.

(No-3 elements in the selected subset are consecutive)

Now,example Array:- {a[1],a[2],.....a[N]}

Say, we calculated dp[1],dp[2],dp[3],dp[4],... by pure observation by our eyes ❤

Now, let's say, we have to calculate dp[i] for any index- 'i' , how do I do that ?

Our choices :-

1. Don't include a[i] . It means we do nothing . The previous answer is considered as the best answer. Hence, dp[i]=dp[i-1].

2. Include a[i]. Imagine this:{a[i-3],a[i-2],a[i-1],a[i]....}.We know that ……..a[i-2],a[i-1] and a[i] are not allowed to be consecutive.So if we select a[i], then the next element we gotta select is a[i-2](dp[i-2]), not ……..a[i-1].Hence.. dp[i]=a[i]+dp[i-2](dp[i-2] →best answer till index …'i-2' of the array)
3. Include a[i].Imagine this:{a[i-3],a[i-2],a[i-1],a[i]....}.We know that ……..a[i-2],a[i-1] and a[i] are not allowed to be consecutive.So if we select a[i], then if we select a[i-1],we are not allowed to select ……..a[i-2]. We will select…a[i-3](dp[i-3]).....Hence.. ....dp[i]=a[i]+a[i-1]+dp[i-3](dp[i-3] →best answer till index …'i-3' of the array)

Maximum of the above three choices will be the answer for "dp[i]"

Voila!We did it!

We found the recurrence relation :-

dp[i]=Maximum(dp[i-1],a[i]+dp[i-2],a[i]+a[i-1]+dp[i-3])

Run a for loop and calculate dp[i] for all “i” from 4 to N.

dp[N] will be the final answer :-)

Some C++ pseudo-code:-

```
//calculate dp[1],dp[2],dp[3].....by observation!!

i=4;

while(i≤N)

{

dp[i]=Maximum(dp[i-1],a[i]+dp[i-2],a[i]+a[i-1]+dp[i-3]);

i++;

}
```

cout<<dp[N];

Problem-2:

We start at index- ‘1’ of the array and strictly end our journey at index- ’N’ . We can make jumps of length “1” or “2” . We have to find a journey with maximum sum.

Example Array :-

{1,-1,100,-11,200,300}

Our Optimal(Best) Journey :- 1 — ->(jump of length-2)100 — ->(jump of length-2)200 →(jump of length-1)300

Hence, Maximum Sum=1+100+200+300=601.

I taught you few tricks earlier and they are as follows:-

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-'i' of the array.
2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...
3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].
4. dp[n] means the answer for the whole array.

Given Example Array:- [1,-1,2,4,3,-5,6,7,8,-80]

dp[i]=Best answer I can get till index- 'i' if I include a[i].

dp[1]=1.

dp[2]=1+(-1)=0

dp[3]=1+2=3(Best answer I can get till index- '3' if I include a[3],where a[3]=2).

How do, we calculate dp[4] , now ??

Think…Think….Think!!!!

We know that we have to include a[4], as per the definition of dp[i].

Hence, dp[4]=a[4]+ ?

We can make jumps of length- “1” OR “2”, so we can reach index- ‘4’ from index- ‘3’(3 — ->4) or index- ‘2’(2 — ->4)

Hence,

dp[4]=a[4]+dp[3]

OR

dp[4]=a[4]+dp[2]

Maximum of both is our answer :-)

Hence, dp[4]=Maximum(a[4]+dp[3],a[4]+dp[2])

The recurrence relation is:
dp[i]=Maximum(a[i]+dp[i-1],a[i]+dp[i-2])

Run a for loop and calculate dp[i] for all “i” from 4 to N.

dp[N] will be the final answer :-)

Some C++ pseudo-code:-

//calculate dp[1],dp[2],dp[3].....by observation!!

i=4;

while(i≤N)

```
{ dp[i]=Maximum(a[i]+dp[i-1],a[i]+dp[i-2]);

i++;

}

cout<<dp[N];
```

Problem-3:

We start at index- ‘1’ of the array and strictly end our journey at index- ’N’ .We can make jumps of length “1” or “2” or “3” or….till “K” . We have to find a journey with maximum sum.

Its same as the problem-2 only difference is that we can make jumps of all lengths from “1……TO…..K” ,

So,our new recurrence relation will be : —

dp[i]=Maximum(a[i]+dp[i-1],a[i]+dp[i-2],a[i]+dp[i-3]+.....a[i]+dp[i-4]+.....+a[i]+dp[i-K])

Run 2 for loops and calculate dp[i] for all “i” till N.

dp[N] will be the final answer :-)

Code:-

```
//dp[1],dp[2],dp[3]...calculate on your own by observation!!

i=4;

while(i≤N)

{

//dp[i]=Maximum(a[i]+dp[i-1],a[i]+dp[i-2],a[i]+.....);
```

```
j=i-1;

g=1;

c=-1000000000000;//just declaring a very small possible
number..

while(j≥1 && g≤k)

{

x=a[i]+dp[i-g];//comparing all of them.....

if(x>c)

{

c=x;
```

```
}

g++;

j- -;

}

dp[i]=c;

i++;

}

cout<<dp[N];
```

One of the problems that will be discussed in the next tutorial is:-

Same as problem-2, how many journeys are possible which will give an even sum or maybe odd sum ;) ? :-)

Diving Deep in Dynamic-Programming with Karan Gujar :-)

P-4:-

We are given four arrays of size-'N' . At each index-'i' , we have to select "1" element out of the 4 given elements,with a special condition that if you select the number from say, j-th array at index-'i' ,then you can't select an element from j-th array again at index-'i+1' , can you maximize the sum ?

Let's take a deep-breath here...!!

Let N=5, Given-4-Arrays are as follows:-

Array(1):- 5,8,-9,2,1(a1[N])

Array(2):-0,0,0,0,-1(a2[N])

Array(3):-1,1,1,-1,1(a3[N])

Array(4):-0,0,0,100,-1000(a4[n])

Solution:-

Make sure you have read my previous-4-tutorials on dynamic-programming to understand this one[Links mentioned above!] .. :-)

Select, ‘1’ from Array-3(index-1),

‘8’ from Array-1(index-2),

‘1’ from Array-3(index-3),

‘100’ from Array-4(index-4),

‘1’ from Array-1(index-5). =[1+8+1+100+1]=111(Maximum possible sum) .

I taught you few tricks earlier and they are as follows:-

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-’i’ of the array.
2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],….etc…
3. We run a for loop, which calculates dp[1],dp[2]…..and finally dp[n].
4. dp[n] means the answer for the whole array. So lets create 4-dp-arrays, dp1[n],dp2[n],dp3[n],dp4[n] :-)

dp1[i] means the best maximum sum we can get if I include element from the first array for the i’th index.

dp2[i] means the best maximum sum we can get if I include element from the second array for the i'th index.

dp3[i] means the best maximum sum we can get if I include element from the third array for the i'th index.

dp4[i] means the best maximum sum we can get if I include element from the fourth array for the i'th index.

For the given 4 arrays to us,

dp1[1]=5=a1[1]

dp2[1]=0=a2[1]

dp3[1]=1=a3[1]

dp4[1]=0=a4[1]

You guys agree?

Let's go in deep thinking mode now....

What will be dp1[2] ?

We will include the element from the first array, no doubt about it.

So, dp1[2]=a1[2]+?

I cannot include a1[1] as it is not allowed, but I can include a2[1] OR a3[1] OR a4[1].

SO I WILL INCLUDE THE LARGEST OF THEM,SO: →

dp1[2]=a1[2]+Maximum(a2[1],a3[1],a4[1])
=a1[2]+Maximum(dp2[1],dp3[1],dp4[1]).

Similarly, dp2[2]=a2[2]+Maximum(dp1[1],dp3[1],dp4[1]).
dp3[2]=a3[2]+Maximum(dp1[1],dp2[1],dp4[1]).

dp4[2]=a4[2]+Maximum(dp1[1],dp2[1],dp3[1]).

WE DID IT, WE FREAKING DID IT!!

We can calculate dp1[N],dp2[N],dp3[N] and dp4[N](just run a for loop till “N” and apply the formula we found!!)

Some C++ psuedo-code:-

```
//Calculate dp1[1],dp2[1],dp3[1],dp4[1]...by yourself!!

i=2;

while(i≤N)

{
```

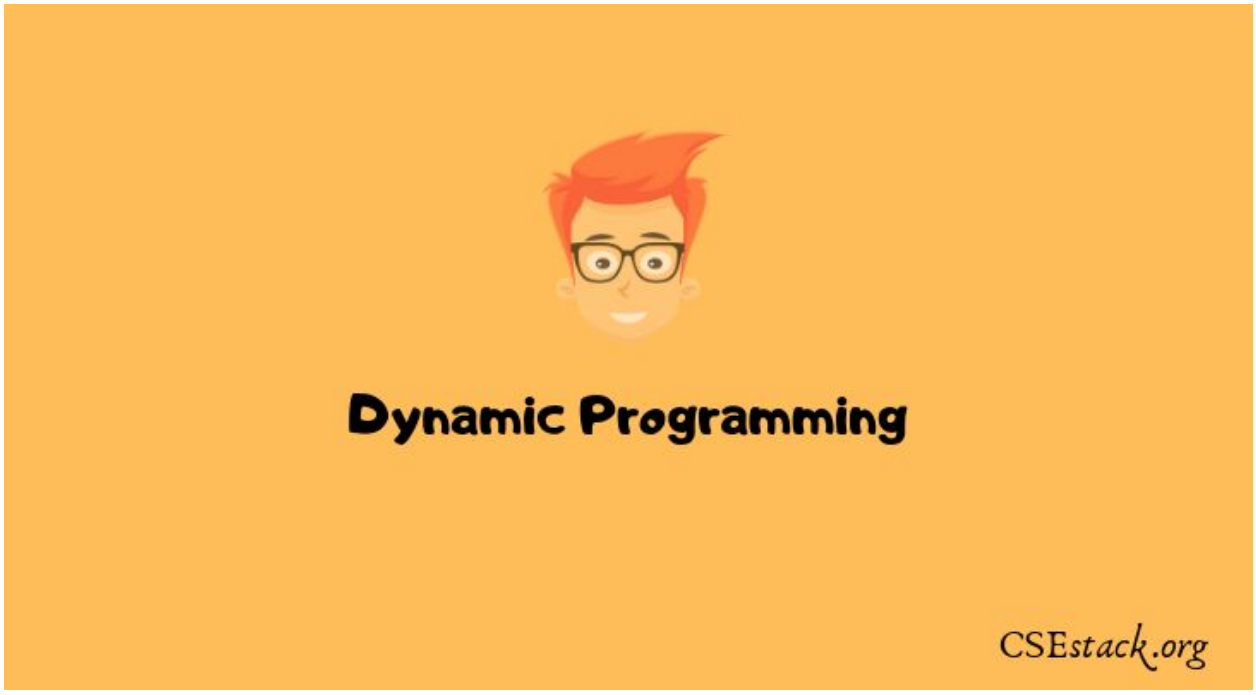
```
dp1[i]=a1[i]+Maximum(dp1[i-1],dp2[i-1],dp3[i-1]);
dp2[i]=a2[2]+Maximum(dp1[i-1],dp3[i-1],dp4[i-1]);
dp3[i]=a3[2]+Maximum(dp1[i-1],dp2[i-1],dp4[i-1]);
dp4[i]=a4[i]+Maximum(dp1[i-1],dp2[i-1],dp3[i-1]);

i++;

}

cout<<Maximum(dp1[N],dp2[N],dp3[N],dp4[N]);
```

We did it today! We won today!

Related-Problems : -

1)https://atcoder.jp/contests/dp/tasks/dp_a

2)https://atcoder.jp/contests/dp/tasks/dp_c


Dynamic Programming
CSEstack.org

“My only job is:- To make you fall in love with Dynamic-Programming,to make you learn this topic in a unique way!!!!”

Happy New Year Guys :)

Traditional-Stuff: “Longest-Increasing-Subsequence”

Problem-Statement:-

p-1:Given an array of “N” integers, find a subset(largest-one) which is strictly increasing.

Example-Array:- {2,1,3,0,5,7}

Largest-Subset which is strictly increasing:- {2,3,5,7}

Its length:-4.

Agreed ?

Got it ?

The rules which I told you to follow in every tutorial :-

1. Usually, we create a dp-array ,and dp[i] means the maximum sum(OR best-answer) we could get till index-'i' of the array.
2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...
3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].
4. dp[n] means the answer for the whole array.

So in our case, dp[i] means the(size-of) longest increasing subset that we could get if the size of array was "i" and the last element of the increasing subset was a[i].

Clearly, max(dp[1],dp[2],.....dp[n]) will be our final answer.

You all agree ?

To know more about this concept, watch my special video on dynamic-programming here:-
https://www.youtube.com/watch?v=Y4AK8Q7_wcM

Say, our given array is :-

{2,1,3,0,5,7}

dp[1]=1....{2}...[Answer to the problem if the size of array was “1” and last-element was “2”]

dp[2]=1....{1}...[Answer to the problem if the size of array was “2” and last-element was “1”]

dp[3]=2....{2,3}.....[Answer to the problem if the size of array was “3” and last-element was “3”]

dp[4]=1....{0}.....[Answer to the problem if the size of array was "4" and last-element was "0"]

Now, guys, lets calculate dp[5] in some unique way!

We know , that a[5]=5.

So, any element like →a[1],a[2] or a[3] or a[4], if it is less than "5", then it can join with "5" to make an increasing subset. Agree ?

Example:-

a[5] can be joined with a[1] and it will look like this after joining:-

{a[1],a[5]}={2,5}(2<5)

Say, for any general-case, a[5] can be joined with a[x],

where, a[5]>a[x] and 1≤x≤4

Agree?

The longest increasing subset till index “5” of array= 1 + The longest increasing subset till index “x” of array.

So, dp[5]=1+dp[x].

How??

dp[5]=subset-form===>{5,{longest-increasing-subset-till-a[x]}}

===>{a[5],{a[x],.....}}

===>1+{a[x],.....}

===>1+dp[x]

We did it!We did it!

Our recurrence-relation is:-

dp[i]=1+dp[x],

where, x=1,2,3,4,....i-1. and also, a[i] should be greater than a[x], or else, it's not valid!

We will run 1-for-loop-to-calculate dp[i] for each 'i' from "1" to "N",

Now, for each, dp[i],

formula is:-

dp[i]=maximum(1+dp[1],1+dp[2],.....1+dp[i-1])

We will need one more for-loop to do this :-)

So,

dp[5]=maximum(1+dp[4],1+dp[3],1+dp[2],1+dp[1])

=maximum(1+1,1+2,1+1,1+1)

=3

Some C++ Pseudo-code:-

dp[1]=1;//__weKnowIt!!

i=2;

while(i≤n)

{

dp[i]=1;//this_is_always_possible_if we only consider {a[i]} as our longest increasing subset.

```
j=1;

while(j≤i-1)

{

if(a[i]>a[j])

{

dp[i]=max(dp[i],1+dp[j]);

}

j++;

}

i++;
```

}

cout<<max(dp[1],dp[2],.....dp[n]);//answer will be maximum of all these {dp-states} as the best sequence might end with a[1] or a[2] or a[3] or ..... A[n], so we find the max() of all!

We did it!

We did it!

P-2:-Given an array of “N” integers, find a subset(largest-one) which is strictly increasing.Also, any 2-consecutive-elements in the subset should differ each other only by one(1).

Say, our given array is :-

{2,1,3,0,5,7,8,1,2}

Answer:- {5,7,8}==>3

If your answer was:- {2,3,5,7,8} . Then its not valid!

Why?

Because “3” and “5” are consecutive in our selected subset and they both differ by “2”(5–3=2)….(Ram-Ram :-))

We only allow consecutive elements to differ by exactly one as mentioned in the problem statement!

Solution is same as P-1!

Eureka moment, right??!!??!!

All you gotta do is make a small change in the if condition of the code :-

if(a[i]>a[j] && a[i]=a[j]+1)

{

//..Statements!!

}

(a[i]=a[j]+1)This makes sure that only elements with difference “1” are selected!!!

Complete pseudo C++ code:-

dp[1]=1;//__weKnowIt!!

i=2;

while(i≤n)

{

```
dp[i]=1;

j=1;

while(j≤i-1)

{

if(a[i]>a[j] && a[i]=a[j]+1)

{

dp[i]=max(dp[i],1+dp[j]);

}

j++;

}
```

```
i++;

}

cout<<max(dp[1],dp[2],...dp[n]) ;
```

We did it!

We did it!

P-3:-Given an array of “N” integers, find a subset(largest-one) which is strictly increasing.Also, any 2-consecutive-elements in the subset should differ each other only by 3 or 5(Nice!).[High-Level-Dp-problem!!]

Think…

Think….

What will be the answer in this case ?

All we gotta do is that make this once change in the if-condition:-

if(a[i]>a[j] && (a[i]=a[j]+3 OR a[i]=a[j]+5))

{

//..Statements!!

}

(a[i]=a[j]+3 OR a[i]=a[j]+5)This makes sure that only elements with difference “3” OR “5” are selected!!!

Complete pseudo C++ code:-

dp[1]=1;//__weKnowIt!!

```
i=2;

while(i≤n)

{

dp[i]=1;

j=1;

while(j≤i-1)

{

if(a[i]>a[j] && (a[i]=a[j]+3 || a[i]=a[j]+5)

{

dp[i]=max(dp[i],1+dp[j]);
```

}

j++;

}

i++;

}

cout<<max(dp[1],dp[2],dp[3],.....dp[n]);

WE DID IT YET AGAIN!!

Final-Problem:-

Find the longest - decreasing - sub - sequence (Subset,I mean :P) in an array.

(Think…think…..)

I’m sure by my teachings, you would have solved this one by now :)

Solution:-

```
dp[1]=1;//__weKnowIt!!

i=2;

while(i≤n)

{

dp[i]=1;

j=1;
```

```
while(j≤i-1)

{

if(a[j]>a[i])

{

dp[i]=max(dp[i],1+dp[j]);

}

j++;

}

i++;

}
```

cout<<max(dp[1],dp[2],.....dp[n]);

Can you spot the only 1-condition I changed?

Yet, we did it again :-)

“There is no pleasure to the peace that can be found in one’s own heart”-Buddha

Keep practicing problems on these type of problems, slowly but surely, the difficulty and complexity of tutorials will rise :-)

Related-Problems:-

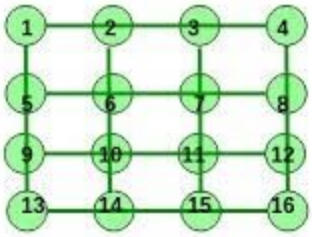
https://stackoverflow.com/questions/52281888/find-longest-inc

reasing-subsequence-with-adjacent-elements-having-a-difference-0

Dp-Tutorial-7 is out !!

Party time. Celebration Time !!

Grid representation



Graph Representation

So how is life going on ?

Let’s get started with part-7 .

And mark my words, “You’ll be a master in DP if you really want it to happen”

“Nothing can stop you!”

I said in the 5th tutorial:-

How many journeys are possible which will give you an even sum ;) ?

A journey is described as : You start at index- ‘1’ of the array and reach at index- ’N’ of the array. At each point, you are allowed to make jumps of length- ‘1’ or ‘2’ .

Example-Array:- {3,3,7,2,3,4,5}

One of the possible-journey:- {3 →(jump of size-2)7 →(jump of size-2)3 →(jump of size-1)4 →(jump of size-1)5}

So how many journeys are possible which will give an even sum ?

Let’s do it with DP :-)

The rules which I keep mentioning in every tutorial :-

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-'i' of the array.
2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...
3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].
4. dp[n] means the answer for the whole array.

Now, lets define dp[i] as :-

"dp[i] means number of journeys which will give an even sum if you start at index- '1' and stop at index- 'i' "

Also,lets define dp1[i] as:-

"dp1[i] means number of journeys which will give an odd sum if you start at index- '1' and stop at index- 'i' "

Also, remember this:-

Odd-Number+Odd-Number=Even-Number

Odd-Number+Even-Number=Odd-Number

Even-Number+Even-Number=Even-Number

Our given array is :-

{3,3,7,2,3,4,5}

We know , that , a[1]=3 , Right??!!

Let’s have fun,maah boiis and girls :-)

dp[1]=0 and dp1[1]=1

Agree?

As the first number (a[1]=3) is odd,we have no ways to make an even-sum journey which starts and ends at index- “1” of the array. So, dp[1]=0.

Also,

As the first number (a[1]=3) is odd,we have 1-WAY to make an odd-sum journey which starts and ends at index- “1” of the array. So, dp1[1]=1.That way is {3}

Lets focus on dp[2] & dp1[2].

dp[2]=1.(Why?, what is that one way?!!)

It is:- {3 — ->(jump of size-1)3}=(3+3)6=even-sum-journey.

dp1[2]=0.(There is no way to get an odd sum if you start at index- “1” and end at index- “2”)

For any ‘i’,

if, a[i] →even,

it means, a[i]can combine with an even sum journey to give an even sum journey again!!

As(EVEN+EVEN=====>EVEN)

So, dp[i]=dp[i-1]+dp[i-2](How?)

If I am at dp[i-1], I can make jump of length- ‘1’ to reach dp[i], and if I am at dp[i-2], I can make jump of length- “2” and reach dp[i], as mentioned in the problem-statement.

Only jumps of length- “1” and “2” are allowed.

In other words, if dp[6]=4 and dp[7]=8(number of even sum journeys till index-6=4 & number of even sum journeys till index-7=8),

(So,you make a jump from index-6 to index-8, so if their were ‘x’ ways to reach- ‘6’, then there will be ‘x’-ways to reach ‘8’ as well, by making a jump.

If you make a jump from index-7 to index-8, so if their were ‘y’ ways to reach- ‘7’, then there will be ‘y’-ways to reach ‘8’ as well, by making a jump.

So, total number of ways=(x+y) [As you can reach “8” from “6” as well as “7”]

)

So, dp[8]=dp[6]+dp[7]=12

What if a[i] is odd ?

Then a[i] can combine with odd-sum-journeys from the past to become an even-sum-journey!!!

As Odd+Odd===>Even-Number.

So, dp[i]=dp1[i-1]+dp1[i-2](How?)

If I am at dp1[i-1], I can make jump of length- ‘1’ to reach dp[i], and if I am at dp1[i-2], I can make jump of length- “2” and reach dp[i], as mentioned in the problem-statement.

Only jumps of length- “1” and “2” are allowed.

In other words, if dp1[6]=14 and dp1[7]=42(number of odd-sum journeys till index-6=14 & number of odd-sum journeys till index-7=42),

(So,you make a jump from index-6 to index-8, so if there were ‘x’ ways to reach- ‘6’, then there will be ‘x’-ways to reach ‘8’ as well, by making a jump.

If you make a jump from index-7 to index-8, so if there were ‘y’ ways to reach- ‘7’, then there will be ‘y’-ways to reach ‘8’ as well, by making a jump.

So, total number of ways=(x+y) [As you can reach “8” from “6” as well as “7”]

)

dp[8]=dp1[6]+dp1[7]=56

So, final recurrence-relation is:-

if(a[i] — ->even)

dp[i]=dp[i-1]+dp[i-2]

if(a[i] — ->odd)

dp[i]=dp1[i-1]+dp1[i-2]

In a similar way, we can calculate, dp1[i], when a[i] is odd or even.

Some-pseudo-C++-code:-

//dp1[1],dp1[2],dp[1],dp[2]....can be calculated just by observation..

i=3;

while(i≤N)

{

if(a[i] — ->even)

dp[i]=dp[i-1]+dp[i-2]

if(a[i] — ->odd)

dp[i]=dp1[i-1]+dp1[i-2]

if(a[i] — ->even)

dp1[i]=dp1[i-1]+dp1[i-2]

if(a[i] — ->odd)

dp1[i]=dp[i-1]+dp[i-2]

i++;

}

cout<<dp[n];

Woohoo!!We did it!

Complete code can be found-here:-https://ideone.com/cpHlpo

HARD-DP-PROBLEM :-

How many journeys are possible which will give you an even sum ;) ?

But, you can make jumps of length- ‘1’, ‘2’ , ‘3’ and ‘4’ :-))

Same-idea as above :)

Only change :- You can reach dp[i] from dp[i-1] or dp[i-2] or dp[i-3] or

dp[i-4] :-)

Just make small changes to the recurrence relation and we are done with a hard-dp-problem!

Some code:-

```
i=5;

while(i≤N)

{

if(a[i] — ->even)

dp[i]=dp[i-1]+dp[i-2]+dp[i-3]+dp[i-4]

if(a[i] — ->odd)

dp[i]=dp1[i-1]+dp1[i-2]+dp1[i-3]+dp1[i-4]

if(a[i] — ->even)

dp1[i]=dp1[i-1]+dp1[i-2]+dp1[i-3]+dp1[i-4]
```

if(a[i] — ->odd)

dp1[i]=dp[i-1]+dp[i-2]+dp[i-3]+dp[i-4]

i++;

}

cout<<dp[n];

In-next-tutorial, I will make your concepts of multi-dimensional-DP really strong, so be ready for it :-)

[We will start with some traditional problems in Tutorial-8]

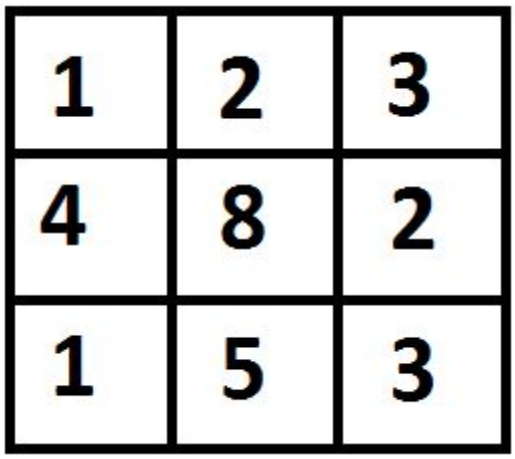
Write your reviews here:-https://docs.google.com/forms/d/1QBfKiqueYUBDsz5wfBswSVNsfasthJjc6dUzzFUsxUc/edit

FALL IN LOVE WITH DP. LOVE IT MORE THAN YOUR GIRLFRIEND(IF YOU HAVE ONE:P)

Related-problems:-1) https://atcoder.jp/contests/dp/tasks/dp_b

2)https://www.hackerearth.com/practice/algorithms/dynamic-programming/introduction-to-dynamic-programming-1/practice-problems/algorithm/odd-even-subarrays-72ad69db/

This is a Matrix (3*3-sized-matrix)given to us.



Let its name be “A”. So,

A[1][1]=1 ;

A[1][2]=2 ;

A[1][3]=3 ;

A[2][1]=4;

A[2][2]=8 ;

A[2][3]=2 ;

A[3][1]= 1;

A[3][2]= 5;

A[3][3] = 3.

[This problem can also be solved for “n*m” sized-matrix with “n”-rows and “m”-columns]

So, if I am at co - ordinate (1,1) …. How can I reach co-ordinate (3,3) if I am allowed to go only down in 1-step or to go only right in 1-step.

Some ways to reach (3,3) from (1,1) :

Way-1:
(1,1)->[Right](1,2)->[Right](1,3)->[Down](2,3)->[Down](3,3)

[Path-Sum:- A[1,1]+A[1,2]+A[1,3]+A[2,3]+A[3,3]=11]

Way-2:
(1,1)->[Right](1,2)->[Down](2,2)->[Down](3,2)->[Right](3,3)

[PATH-sum: 19]

Way-3: ....

Way-4: ...

....

And some more ways exist!

So the original-question is :- You start at (1,1) in a matrix and have to reach (3,3) in the matrix ,

which of the way(PATH) will give,

minimum sum(PATH-SUM) ,what will be that minimum-sum ?

The rules which I always taught you : -

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-'i' of the array.
2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...
3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].
4. dp[n] means the answer for the whole array.

Now,we change and manipulate the rules like this :-

“Everything remains the same but, we define dp[i][j](2-Dimensional-Array) which means the best answer till

co - ordinate (i,j) of the matrix.”

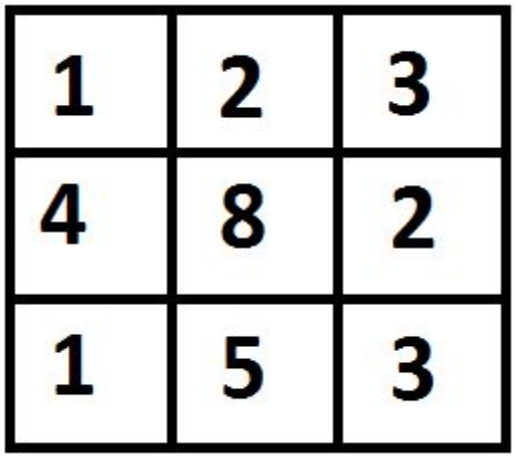
So, in further tutorials,remember, the basic concepts remain the same but dp-array and can be 1-Dimensional-Array,2-Dimensional-Array,3-Dimensional-Array, etc. depending on the needs!

In DP, all you gotta do is :

1. Figure out how many dimensions of DP will be needed.
2. What will be the states of the DP(What will be the meaning of i,j,k,.. for dp[i][j][k]... ?)
3. Recurrence-relation.

Note down these things and remember them perfectly.

Our given-(3*3) matrix is :- (We can solve this problem for any matrix size of n*m, in this case n=3 and m=3)



dp[1][1]=A[1][1]=1[minimum-path-sum to reach (1,1)]

Agree?

dp[1][2]=A[1][1]+A[1][2]=(1+2=3)[[minimum-path-sum to reach (1,2)]]

AGREE ?

dp[1][3]=A[1][1]+A[1][2]+A[1][3]=6[[minimum-path-sum to reach (1,3)]]

dp[2][1]=A[1][1]+A[2][1]=1+4=5.[[minimum-path-sum to reach (2,1)]]

What about dp[2][2] ?

There are 2-ways to reach (2,2)

Way-1:-(1,1) →(2,1)
→(2,2)[Path-Sum:-A[1][1]+A[2][1]+A[2][2]=13]

Way-2:-(1,1) →(1,2)
→(2,2)[Path-Sum:-A[1][1]+A[1][2]+A[2][2]=11]

Which of the both ways has minimum sum ?

WAY-2!!!!!![ITS SUM IS ONLY 11]

Hence, dp[2][2]=11.....Equation-(x)

Now, its the time to derive recurrence-relation guys/girls .

For (2,2) we can reach it through (1,2) or (2,1).

AGREED ?

We can reach (2,2) from (1,2).

If we know the minimum sum till (1,2)[which is dp[1][2]] , we can calculate dp[2][2].

dp[2][2]=A[2][2]+dp[1][2](Best-minimum-sum till (1,2) + Value of co - ordinate (2,2))

Or we can reach (2,2) from (2,1) .

So,

If we know the minimum sum till (2,1)[which is dp[2][1]] , we can calculate dp[2][2].

dp[2][1]=A[2][2]+dp[2][1](Best-minimum-sum till (2,1) + Value of co - ordinate (2,2))

Minimum of both the choices will be the answer , hence ,

dp[2][2]=minimum(a[2][2]+dp[2][1],a[2][2]+dp[1][2])

Hence,dp[2][2]=minimum(8+5,8+3)=11(which is same as equation-(x))

We did it!

Now, substitute, “i” and “j”, we get the recurrence-relation :-

dp[i][j]=minimum(a[i][j]+dp[i][j-1],a[i][j]+dp[i-1][j])

Run 2-for-loops and calculate dp[i][j] using the formula above :-)

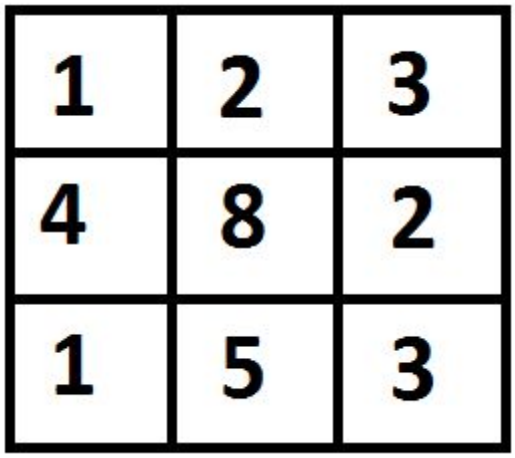
Time-Complexity:- O(n*n)

dp[3][3] or dp[n][m](if matrix is of “n * m” size) will be the final-answer :-)

Some C++-pseudo-code:

```
i=2;

while(i<=n)

{

j=2;
```

```
while(j<=m)

{

dp[i][j]=min(a[i][j]+dp[i-1][j],a[i][j]+dp[i][j-1]);



j++;

}

i++;

}

cout<<dp[n][m];
```



Let its name be “A”. So,

A[1][1]=1 ;

A[1][2]=2 ;

A[1][3]=3 ;

A[2][1]=4;

A[2][2]=8 ;

A[2][3]=2 ;

A[3][1]= 1;

A[3][2]= 5;

A[3][3] = 3.

[This problem can also be solved for “n*m” sized-matrix with “n”-rows and “m”-columns]

So, if I am at co — ordinate (1,1) …. How can I reach co-ordinate (3,3) if you are allowed to :

1. Go Down in 1-step[(i,j) →(i+1,j)]
2. Go Right in 1-step[(i,j) →(i,j+1)]
3. Go diagonally in 1-step[(i,j) →(i+1,j+1)]

Now the real fun of DP-begins :-)

Some ways to reach (3,3) from (1,1) :

Way-1:
(1,1)->[Right](1,2)->[Right](1,3)->[Down](2,3)->[Down](3,3)

[Path-Sum:- A[1,1]+A[1,2]+A[1,3]+A[2,3]+A[3,3]=11]

Way-2:
(1,1)->[Right](1,2)->[Down](2,2)->[Down](3,2)->[Right](3,3)

[PATH-sum: 19]

Way-3: (1,1) →[Diagonal](2,2) →[Right](2,3) →[Down](3,3)

[Path-sum: →14]

Way-4: ...

....

And some more ways exist!

So the original-question is :- You start at (1,1) in a matrix and have to reach (3,3) in the matrix ,

which of the way(PATH) will give,

minimum sum(PATH-SUM) ,what will be that minimum-sum ?

The rules which I always taught you : -

1. Usually, we create a dp-array , and dp[i] means the maximum sum(best-answer) we could get till index-'i' of the array.
2. We need to find a recurrence relation, in simple words, we need to create a formula for dp[i] in terms of dp[i-1],dp[i-2],....etc...
3. We run a for loop, which calculates dp[1],dp[2].....and finally dp[n].
4. dp[n] means the answer for the whole array.

Now,we change and manipulate the rules like this :-

“Everything remains the same but, we define dp[i][j](2-Dimensional-Array) which means the best answer till co — ordinate (i,j) of the matrix.”

So, in further tutorials,remember, the basic concepts remain the same but dp-array and can be 1-Dimensional-Array,2-Dimensional-Array,3-Dimensional-Array, etc. depending on the needs!

In DP, all you gotta do is :

1. Figure out how many dimensions of DP will be needed.
2. What will be the states of the DP(What will be the meaning of i,j,k,.. for dp[i][j][k]... ?)
3. Recurrence-relation.

Our given-(3*3) matrix is :- (We can solve this problem for any matrix size of n*m, in this case n=3 and m=3)

1	2	3
4	8	2
1	5	3

dp[1][1]=A[1][1]=1[minimum-path-sum to reach (1,1)]

Agree?

dp[1][2]=A[1][1]+A[1][2]=(1+2=3)[[minimum-path-sum to reach (1,2)]]

AGREE ?

dp[1][3]=A[1][1]+A[1][2]+A[1][3]=6[[minimum-path-sum to reach (1,3)]]

dp[2][1]=A[1][1]+A[2][1]=1+4=5.[[minimum-path-sum to reach (2,1)]]

What about dp[2][2] ?

There are 3-ways to reach (2,2)

Way-1:-(1,1) →(2,1)
→(2,2)[Path-Sum:-A[1][1]+A[2][1]+A[2][2]=13]

Way-2:-(1,1) →(1,2)
→(2,2)[Path-Sum:-A[1][1]+A[1][2]+A[2][2]=11]

Way-3:-(1,1) →(2,2)[Path-Sum:- A[1][1]+A[2][2]=9]

Which of the both ways has minimum sum ?

WAY-3!!!!!![ITS SUM IS ONLY 9]

Hence,dp[2][2]=9.....Equation-(x)

Now, it's the time to derive recurrence-relation guys/girls .

For (2,2) we can reach it through (1,2) or (2,1).

AGREED ?

We can reach (2,2) from (1,2).

If we know the minimum sum till (1,2)[which is dp[1][2]] , we can calculate dp[2][2].

dp[2][2]=A[2][2]+dp[1][2](Best-minimum-sum till (1,2) + Value of co — ordinate (2,2))

Or we can reach (2,2) from (2,1) .

So,

If we know the minimum sum till (2,1)[which is dp[2][1]] , we can calculate dp[2][2].

dp[2][1]=A[2][2]+dp[2][1](Best-minimum-sum till (2,1) + Value of co — ordinate (2,2))

OR We can reach (2,2) from (1,1).

If we know the minimum sum till (1,1)[which is dp[1][1]] , we can calculate dp[2][2].

dp[2][2]=A[2][2]+dp[1][1](Best-minimum-sum till (1,1) + Value of co — ordinate (2,2))

Minimum of all the three choices will be the answer , hence ,

dp[2][2]=minimum(a[2][2]+dp[2][1],a[2][2]+dp[1][2],a[2][2]+dp[1][1])

Hence,dp[2][2]=minimum(8+5,8+3,8+1)=9(which is same as equation-(x))

We did it!

Now, substitute, “i” and “j”, we get the recurrence-relation :-

dp[i][j]=minimum(a[i][j]+dp[i][j-1],a[i][j]+dp[i-1][j],a[i][j]+dp[i-1][j-1])

Run 2-for-loops and calculate dp[i][j] using the formula above :-)

Time-Complexity:- O(n*n)

dp[3][3] or dp[n][m](if matrix is of “n * m” size) will be the final-answer :-)

Some C++-pseudo-code:

```
i=2;

while(i<=n)

{

j=2;

while(j<=m)

{

dp[i][j]=min(a[i][j]+dp[i-1][j],a[i][j]+dp[i][j-1],a[i][j]+dp[i-1][j-1]
);

j++;

}
```

```
i++;

}

cout<<dp[n][m];
```

Actual code can be accessed from here :-

https://ideone.com/2fH67R

Get ready for tutorial-10!!

Some more possible variations of this problem:-

1)Same question as above but only “down” and “diagonal” moves are allowed.

2)Same question as above but only “right” and “diagonal” moves are allowed.

[Try them on your own, I’ll reveal their answer in next tutorial!!]

Videos:- https://www.youtube.com/channel/UCNMdVHJekNJZpUivMu9mXKA
Related-Problem(s) :- https://codeforces.com/contest/429/problem/B

“Coding is easy if you understand everything clearly, clarity is the true power”

Lets Master DP In A fUN wAY ❤ ❤ ❤ ❤ ❤ ❤ ❤

Let’s have some general talk about DP and its states.

Whenever , we want to solve a DP-problem, we make a recurrence relation, run the for loop(s) and solve the problem.

Example of recurrence-relation :

dp[i][j][k]=dp[i-1][j+1][k-4]+.....+....

We call ‘i’ , ‘j’ , ‘k’ as the states of the DP .

All we have to do is :

1)Identify the states of DP and make a recurrence relation, then we are done!

Make sure you are thorough with the previous “9” tutorials of mine before we move ahead.

Dynamic-Programming can be broadly divided into the following types : —

1. One-Dimensional DP
2. Two-Dimensional DP : i)Mixt-Type-1 , ii)Mixt-Type-2 ,iii)Grid-Based
3. DP based on counting the number of ways/sequences possible with some property.
4. Digit-DP(Also remainder-DP)
5. Tree-DP
6. DP on D.A.G(Directed-Acyclic-Graph)
7. Miscellaneous/(More-than-2-Dimensions — DP)

It is better to study each type in depth so the problem can be recognized and broken down into similar types very fast !

We have discussed many variations of One-Dimensional-DP, few of Grid-based as well!!!

Let's discuss the following One-Dimensional-DP problem now :-

Problem {1} : Given two arrays , "A" and "B" , for each index, select an element from either "A" or "B" , and the total sum should be maximized.

One special condition to be followed : —> In all the elements we select, we cannot select more than three elements that are consecutive in Array- "A" or Array- "B" .
.....(i)

A:{1,-2,3,4,5,6}

B:{0,-2,0,1,2,5}

[1-Based-indexing is assumed for both the arrays]

We will select : A[1] + B[2] + A[3] + A[4] + A[5] + B[6] = 16(Maximum-Sum)

We cannot select (A[1]+B[2]+A[3]+A[4]+A[5]+A[6]=17), because we selected A[3],A[4],A[5],A[6].....which is more than 3 consecutive elements, which is not allowed according to ..(i)

Let dp1[i] be the array whose meaning is the best answer we can get till index- 'i' if we include A[i] .

Similarly, dp2[i] be the array whose meaning is the best answer we can get till index- 'i' if we include B[i] .

A:{1,-2,3,4,5,6}

B:{0,-2,0,1,2,5}

dp1[1]=A[1]=1

dp2[1]=B[1]=0

dp1[2]=A[2]+Maximum(B[1],A[1])=(-2+Maximum(0,1))=-2+1=-1

dp2[2]=B[2]+Maximum(B[1],A[1])=(-2+Maximum(0,1))=-2+1=-1

dp1[3]=A[3]+Maximum(dp1[2],dp2[2])=2

dp2[3]=B[3]+Maximum(dp1[2],dp2[2])=-1

Now, let's calculate dp1[4] and dp2[4] in a special way :-) {My usual trick I use so that we can make a recurrence relation :-) }

For dp1[4] , what are the choice(s) do I have ?

Choice →1 : Select A[4] and dp2[3]

Choice-2 : Select A[4] , A[3] and dp2[2] (Selection of A[4] and A[3] is done as we are allowed to select two consecutive elements ❤)

Choice →3 : Select A[4] , A[3],A[2] and dp2[1] (Selection of A[4] ,A[3] and A[2] is done as we are allowed to select three consecutive elements ❤)

Choice{4} : — No more choice! (We cannot select more than three consecutive elements :-))

Maximum of all the three choices is our answer !!!!

Hence,

dp1[4] = Maximum(A[4]+dp2[3],A[4]+A[3]+dp2[2],A[4]+A[3]+A[2]+dp2[1])

Similarly,

dp2[4]=

Maximum(B[4]+dp1[3],B[4]+B[3]+dp1[2],B[4]+B[3]+B[2]+dp1[1])

Our recurrence relation is as follows :

[Lets call array “A” as “a1” and array “B” as array “a2”]

$dp1[i]=max(a1[i]+dp2[i-1],a1[i]+a1[i-1]+dp2[i-2],a1[i]+a1[i-1]+a1[i-2]+dp2[i-3])$;

dp2[i]=max(a2[i]+dp1[i-1] , a2[i]+a2[i-1]+dp1[i-2] , a2[i]+

a2[i-1]+a2[i-2]+dp1[i-3]);

We will , as always run a for loop and calculate dp1[n] & dp2[n] .

Maximum(dp1[n],dp2[n]) will be our answer :-)

Some C++ pseudo-code :

```
dp1[1]=a1[1];

dp2[1]=a2[1];

dp1[2]=a1[2]+Maximum(a2[1],a1[1]);

dp2[2]=a2[2]+Maximum(a2[1],a1[1]);

dp1[3]=a1[3]+Maximum(dp1[2],dp2[2]);

dp2[3]=a2[3]+Maximum(dp1[2],dp2[2]);

for(i=4;i<=n;i++)

{
```

$dp1[i]=max(a1[i]+dp2[i-1],a1[i]+a1[i-1]+dp2[i-2],a1[i]+a1[i-1]+a1[i-2]+dp2[i-3]);$

$dp2[i]=max(a2[i]+dp1[i-1],a2[i]+a2[i-1]+dp1[i-2],a2[i]+a2[i-1]+a2[i-2]+dp1[i-3]);$

}

$cout<<max(dp1[n],dp2[n]);$

Done :-)

Let's talk about 2-Dimensional-DP , in particular, mixt-type-1 .

In this type of DP, we calculate DP[i][j] , which means answer for the range [i.....j] of the array .

Example Array : {1,5,2,3,3,4,5,6,6,6,6,6}

DP[2][5]= answer for the range {a[2].....a[5]}= { 5,2,3,3}

First we calculate the answer for small ranges like , DP[1][1],DP[2][2],.....then using these answers , we calculate the answer for larger ranges like DP[1][80],etc.

If the size of the array is “N”, obviously DP[1][N] , gives the answer for full range :-)

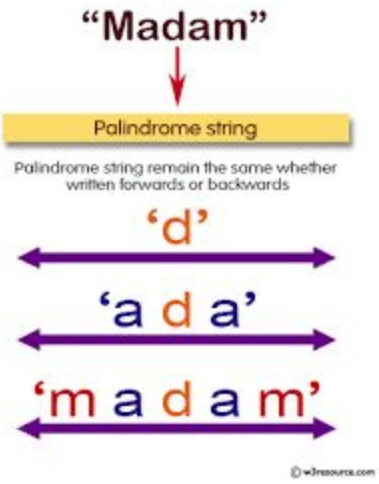
Problem[2] :

Given a string ; find the length of the longest palindromic sub — string :-))

Example STRING : — abbbb

ANSWER :Length of longest palindromic-substring : (4) {bbbb===>PALINDROME)

So, first, we will calculate answer for all substrings of length- “1” ,



Defining our dp as,

dp[i][j]=0 {susbstring from "i" to "j" is NOT A PALINDROME}

dp[i][j]=1 {susbstring from "i" to "j" is A PALINDROME}

Example:- cabbad

dp[1][1]=1...(“c” is a palindrome)

dp[2][2]=1...(“a” is a palindrome)

dp[3][3]=1...(“b” is a palindrome)

dp[4][4]=1...(“b” is a palindrome)

dp[5][5]=1...(“a” is a palindrome)

dp[6][6]=1...(“d” is a palindrome)

Now, lets do the same for all sub-strings of length- “2” :

dp[1][2]=0...(“ca” is not a palindrome)

dp[2][3]=0...(“ab” is not a palindrome)

dp[3][4]=1...(“bb” is a palindrome)

dp[4][5]=0...(“ba” is not a palindrome)

dp[5][6]=0...(“ad” is a palindrome)

Now, lets do the same for all sub-strings of length-”3" but in a unique fashion so we can find a recurrence relation {My ususal trick so you guys reach the recurrence relation smoothly! ❤ ❤ ❤ ❤ ❤ ❤ }

Lets say , we have to calculate dp[1][3] ....

if (s[1]==s[3] And s[2] →Palindrome) ,

Then, dp[1][3]=1{Palindrome} ,

else,

NO, dp[1][3]=0.

{Note:- dp[1][3]=0, as s[1]!=s[3])

In general, lets calculate dp[3][456] , it simply means if the substring s[3.....456] , palindrome or not ?

It is a palindrome only if s[3]===s[456] and s[4.....455] is a palindrome , you guys agree ?

So, our recurrence relation is :

{We fill small value like dp[1][1],dp[2][2],.....and dp[1][2],dp[2][3],...by our own and do the following for the rest } : —

if(s[i]==s[j] && dp[i+1][j-1]==1)

{

dp[i][j]=1;

}

else

{

dp[i][j]=0;

}

As simple as that !

Some C++ Pseudo-Code :-

max=0;

j1=3;

while(j1≤n)

```
{
i=1;
j=j1;
while(i≤n-j1+1)
{
if(s[i]==s[j] && dp[i+1][j-1]==1)
{
dp[i][j]=1;
max=j-i+1;
}
```

```
else

{

dp[i][j]=0;

}

j++;

i++;

}

j1++;

}
```

We will meet again soon ❤ ❤ ❤ ❤ ❤ ❤ ❤