Dynamic Programming:

1. Overlapping sub problem
2. Optimal (max min etc.) Substructure

* Two approaches

1. Bottom up, Top down(memorize)

* Thought process.

1. First try to write recursive formula.
2. Check for case 0 etc. (this will decide the length of tabulation matrix)
3. Similar(below) 1-2, 3-7, 8-10, 11-14, 15-17(egg drop without len), 18, 19-20, 21,22, 23-24
4. Pattern
   1. LIS: 1D
   2. LCS: 2D
   3. Loop to iterate over len

# Fibonacci

* 1. Recursion

Fib(n)

{

If (n <=1) //base

Return n;

Return fib(n-1) + fib(n-2);

}

* 1. Bottom up

// Create array N and init DP[0] = 0 , DP[1] = 1

For I : 2-> n

DP[i] = DP[i-1] + DP[i-2];

Return DP[n]

* 1. Top Down(memorize)

// Create array N and init to - 1

Fib(n)

{

If (DP[n] != -1) // memory

Return DP[n];

If (n <=1) //base Cond

DP[n] = n;

Return DP[n];

DP[n] = fib(n-1)+ fib(n-2);

Return DP[n];

}

# Binomial coefficients // 2D version of Fib

BC(n,k) = BC(n-1, k-1) + BC(n-1,k) //select last ele or leave last ele

BC(n,0) = BC(n,n) = 1

// DP[n+1][n+1] , use above formula to fill

I: 0 to n

J : 0 to **min(i, k)**

# Longest Increasing Subsequence LIS

1. Bottom up: input is Arr[]

LIS(int Arr[], int n)

{

DP[n], init all to 1

For I :1->n-1

For j: 0 to i-1

If (Arr[j] < Arr[i] and DP[i] < DP[j] + 1)

DP[i] = DP[j] +1;

}

//Iterate over A and find the max

// To print the seq, traverse back on the DP array

// To print the seq, take a array of struct and put previous max ele pos also there

# Longest increasing sum sub seq

LISS(int arr[], int n)

{

//DP[n] , init to arr element

For(i=0; i< n ; i++)

For(j = 0; j< I; j++)

If( arr[i] > arr[j] && DP[j] + DP[i] > DP[i])

DP[i] = DP[i] + DP[j];

//iterate over DP and find Max

}

# Longest bitonic subsequence

LBS = LIS + LDS -1 (longest increasing + longest decreasing -1)

Three loop , first to find LIS, second to LDS and third to find max (LIS[i] + LDS[j] -1]

# Maximum length chain of pairs

LCOP(int arr[], int n)

{

//DP[n] , init to 1

For(i=0; i< n ; i++)

For(j = 0; j< I; j++)

If( arr[i].a > arr[j]. b && DP[j] + 1 > DP[i])

DP[i] = 1 + DP[j];

//iterate over DP and find Max

}

# Box Stacking Problem

* Create an arr[3n], by exchanging l, b,h
* Sort this new array with decreasing area(l\*b)
* Apply lis

For(I = 0; I < 3n; i++)

For (j= 0 ; j < I ; j++)

If(arr[i].l < arr[j].l && arr[i].b < arr[j].b && DP[i] < DP[j] + arr[i].h)

DP[i] = DP[j] + arr[i].h

* Find max and return

# Longest common subseq

//L[m +1][n+1] , init to 0

For i: 1-> m

For j :1->n

If(a[i-1] = b[j-1])

L[i][j] = 1 + l[i-1][j-1]

Else

L[i][j] = max(L[i][j-1], L[i-1][j])

//l[m-1][n-1] is answer

# Edit distance.

Input a[m] b[n], operation insert, delete, replace

//L[m +1][n+1]

For i: 0-> m

For j :0->n

If (i==0)

L[i][j] = j

If(j == 0)

L[i][j] = i

If(a[i-1] = b[j-1])

L[i][j] = L[i-1][j-1]

Else

L[i][j] =1 + min(L[i][j-1], L[i-1][j], L[i-1][j-1])

//L[m][n] will have answer

# Min Cost path

Input: a[m][n], value is cost

Reach from 0,0 to m,n with min cost path

//m[M][N] , INIT 0,0 to a[0][0]

// row and col, min travel is through straight line

For i: 1-> n-1

M[0][i] = m[0][i-1] + a[0][i]

For j: 1-> m-1

M[j][0] = m[j-1][0] + a[j][0]

For i: 1-> m-1

For j :1->n-1

M[i][j] = a[i][j] + min (m[i-1][j], m[i-1][j-1], m[i][j-1])

//ans will be m[m-1][n-1]

# Coin change

Input: a [m], having list of coin denomination, we have infinite supply of each coin and N

Output: no of ways the coin change can be made

Recursive

{

If n =0, return 0, if m = 0 return 1

Cc(n,m) = cc(n-1,m) + cc(n, m-a[n-1])

}

CC(A[], n,m)

{

DP[n+1][m+1];

// fill first row with 0

//fill 1st col with 1

For (i:1 to n)

For(j: 1 to m)

DP[i][j] = DP[i-1][j] + DP[i][j-a[i-1]

}

# 0-1 Knapsack problem:

1. (Top down) , memorize

Create a matrix of I \* w,init to -1

Knapsack(v[], w[], W, I)

{

If (I < 0 || W == 0) // base condition

Return 0;

If a[i][W] != -1

Return A[i][W]

If w[i] > W

A[i][W] = knapsack(v, w, W, i-1); // not include

Else

A[i][W] = max(v[i-1]+ knapsack(v, w, W-w[i], i-1), knapsack(v, w, W, i-1)); // include,not

Return A[i][W]

}

1. Bottom UP

Create table n+1 \* W + 1, init 1st row and col to 0

For i: 1-> n

J : 1 -> W

If (w[I -1] <= j)

A[i][j] = max(A[i-1][j], v[i-1] + a[i-1][j-w[i-1]]);

Else

A[i][j] = a[i-1][j]

// search the matrix to find max

# Subset sum

SS(a[], n, sum)

{

Bool DP[n+1][sum+1]

// if sum == 0 , col 1 true

// if n == 0 and sum != 0 , first row false

For (I : 1 to n+1)

For(j : 1 to sum+1)

DP[i][j] = DP[i-1][j] | DP[i-1][j-a[i-1])

}

# Partition SUM: similar to sum subset, devide sum /2

Sum should be even

# EGG DROP Prob // no len

If E = 1, return F

If k = 1 return 1

We can omit 0 col and 0 row

If E = 0 , return 0

If F= 0, return 0

Two case , either egg breaks or survive on kth floor

EDP(E,F) = 1 + max(EDP(E-1, K-1), EDP(E, F-K)), find min for k :0 to f

EDP(E,F)

{

DP[E+1][f+1]

//fill first row with 0

// first col 0

// second row with j

// sec col with 1

For ( I : 2 to E)

For (j : 2 to F)

DP[i][j] = max

For (k : 1 to j)

Temp = 1+ max (DP [i-1][k-1], DP[i][j-k])

If (temp < DP[i][j])

DP[i][j] = temp

Return DP[E][F]

}

# Longest Palindromic subseq

LPS(a[], n)

{

DP[n][n]

Fill diag with 1(i==j)

For (len 2 to n)

For (I =0 to n-len + 1)

J = I + j -1

If (a[i] == a[j] and len == 2)

DP[i][j] =2;

If(a[i]==a[j])

DP[i][j] = DP[i+1][j-1] + 2

Else

DP[i][j] = max(DP[i+1][j], DP[i][j-1])

}

# Matrix chain multiplication

// for 1 matrix return 0

For recursive think

MCM(A[], I , j)

MCM(a[], n)

{

DP[n][n]

If(i==j), fill diag with 0

For (len = 2 to n)

For (I : 1 to n-len + 1)

DP[i][j] = max

For (k: I to j)

Temp = DP[i][k] + DP[k][j] + A[i-1]\*a[k]\*A[j]

If (temp< DP[i][j])

DP[i][j] = Temp

Return DP[1][n-1] // start and end

}

18. Palindrome partition: same as matrix chain multiplication, with one extra bool ISPalindrome[n][n] , true if i==j and if a[i] == a[j] and ISPalindrome[i+1][j-1] = true, if a[i] == a[j] and len = 2

# 19. Rod Cutting prob

RC(A[], N) =

{

DP[n+1]

DP[0] = 0

For (I :1 to n+1)

{

DP[i] = min

For(k :0 to i)

DP[i] = max( a[k] + DP[i-k-1], DP[i])

}

}

# Max size sub sq matrix with all 1

MSSS(M[][], n)

{

DP[n][n] , copy M[][]

For(I : 0 to n)

For (j: 0 to n)

If(M[i][j] == 1)

DP[i][j] = min(DP[i-1][j-1], DP[i][j-1], DP[i-1][j]) + 1

Else

0

}

# UGLY Nomber

UN(n)

{

DP[n]

DP[0] = 1

Index2 = index3= index5 = 0

Num2 = index2\*2;

Num3= index3\* 3;

Num5 = index5 \* 5;

For(I : 1 to n)

{

Next = min(num2, num3, num5)

DP[i] = next

If next == num2

{

Index2++;

Num2 = DP[index2]\*2

}

If next == num3

{

Index3++;

Num3 = DP[index3]\*3

}

If next == num5

{

Index5++;

Num5 = DP[index5]\*5

}

}

}

# Number of bin string without consecutive ones

D[n] = nmberendingWith0a[n] + NumEndWith1b[n]

D[1] = 1 + 1, nmberendingWith0 a[1]=1 , NumEndWith1 b[1]= 1

For (I : 2 to n)

B[i] = a[i-1] // 1s can be added here only

A[i] = a[i-1]+ b[i-1] // 0 can be added to both

D[i] = a[i] + b[i]

# LARGEST SUM CONT Subarray

TotalMaxSum, LocalMaxSum,

# Largest Palindromic substring (substring not subseq)

LPS(a[], n)

{

DP[n][n]

DP[i][i] = 1

maxLen = 0

Bool IsPal[n][n]

Len: 2 to n-2

I: 0 to n-len+1

If(a[i] =a[j] and len = 2 )

Start = I , maxlen =2, IsPal[i][j] = true

If(a[i] == a[j] and isPal[i+1][j-1] == true)

Start = I, maxlen = len, IsPal[i][j] = true

}

# Floyd Warshal : all pair shortest path

DP[n][n] = INT\_MAx

FW(m[][], n)

For (I : 0 to n)

For (j : 0 to n)

For(k: 0 to n)

If DP[i][j] < M[i][k] + M[k][j]

DP[i][j] = M[i][k] + M[k][j]

# BELLMAN FORD : shortest path from a src

D[n], init to max for src to 0

For: loop over vertex

For : loop over edge(ex uv)

D[v] > D[u] + uv

D[v] = D[u] + uv

# Not Attempted

Word Wrap

Max number of jumps

Optimal bin search tree

Largest independent set

Max sum rectangle

Boolean parenthesis