dp notes

0/1 Knapsack

```
Input
value[], weight[], CapacityDp statedp[i][j] represents max su
m of value we get by using items from 0 to i and bag of capac
ity j
i = 0 to n
j = 0 to Capacitydp[i][j] =
  dp[i-1][j]      if j < weight[i]
  max(dp[i-i][j], value[i] + dp[i-1][j-weight[i]])    otherwi
se
```

LCS longest common subsequence

```
Input
String s1 and String s2
Dp state
dp[i][j] represents LCS of s1.substring(0,i) and s2.substring
(0,j)
dp[i][0] = dp[0][j] = 0
i = 1 to s1.length
j = 1 to s2.lengthdp[i][j] =
  dp[i-1][j-1]     if s1[i-1] == s2[j-1]
  max(dp[i-1][j], dp[i][j-1])     otherwise
```

LPS longest palindromic subsequence

```
Input
String s
Dp state
dp[i][j] represents longest palindromic subsequence's length
```

```
of substring(i, j), here i, j represent left, right indexes i
n the string
dp[i][i] = 1 // single character palindromeRun matrix loop di
agonally because dp[i][j] depends in dp[i+1][j-1]len = 1 to
s.length
i = 0 to s.length - len
j = i+len-1dp[i][j] =
  2 + dp[i+1][j-1]     if s[i] == s[j]
  max(dp[i+1][j], dp[i][j-1])    otherwise
```

Edit Distance

```
Input
String s1 and String s2
Dp state
dp[i][j] represents minimum number of operations to convert s
1.substring(0,i) to s2.substring(0,j) using replace, insert o
r delete operations.dp[i][0] = i
dp[0][j] = j
i = 1 to s1.length
j = 1 to s2.lengthdp[i][j] =
  dp[i-1][j-1]   if s1[i] == s2[j]
  1 + min(dp[i-1][j-1], dp[i][j-1], dp[i-1][j])// min(replac
e, insert, delete)
```

Regular expression matching

```
Input
text String s and pattern String p.
Dp state
dp[i][j] represents whether s.substring(0,i) matches with p.s
ubstring(0,j). It contains boolean value.dp[0][0] = true
dp[0][j] = dp[0][j-2] if p[j-1] == '*' i = 1 to s.length
j = 1 to p.lengthdp[i][j] =

dp[i-1][j-1]     if s[i] == p[j] or p[j] == '.'
```

```
dp[i][j-2] || (dp[i-1][j] && (s[i] == p[j-1] or p[j-1] ==
'.'))    if        p[j] == '*'

false   otherwise
```

Rod cutting

```
Input
array price[] where price[i] contains price of rod of length
i
Dp state
dp[i] represents max value we can get for rod of length i
i = 1 to n
dp[i] =
 for j in 0 to i-1
   max(dp[i], price[j] + dp[i-j-1]
```

Optimal binary search

```
Input
array frequency[] where frequency[i] represent search frequen
cy of value at i index.
Dp state
dp[i][j] represents minimum cost of search for elements from
index i to j
dp[i][i] = frequency[i]// Another example of diagonally align
ed dp statelen = 2 to s.length
i = 0 to s.length - len
j = i+len-1dp[i][j] =
  for k in i to j
    sum(i, j) + min(dp[i][k-1], dp[k+1][j])
```

Minimum Coin change

```
Input
array coins[] where coins[i] is denomination and total for wh
ich we need to find minimum coin change.Dp statedp[i][j] repr
esents minimum coin change required to create value j using c
oins from 0 to i
i = 0 to coins.length
j = 0 to total
dp[i][j] =
  min(1+dp[i][j - coins[i]], dp[i-1][j])   if j >= coins[i]
  dp[i-1][j]   otherwise
```

Matrix chain multiplication

```
Input
array arr[] representing size of matrix {rows, cols}
Dp state
dp[i][j] represents minimum cost of multiplying matrices from
index i to j// Another example of diagonally aligned dp state
dp[i][i] = 0 // single matrix don't have any cost of multipli
cation
dp[i][i+1] = arr[i].rows * arr[i].cols*arr[i+1].colslen = 3 t
o arr.length
i = 0 to arr.length - len
j = i+len-1dp[i][j] =
  for k in i to j-1
    min(dp[i][k] + dp[k+1][j] + arr[i].rows*arr[k].cols*arr
[j].cols)
```

Subset sum problem

```
Input array nums[] containing n integers and totalSum
Dp state
dp[i][j] represents if using first i numbers from nums array
is it possible to create a subset of totalSum j.dp[0][j] = fa
lse j > 0 not possible to create subset with sum j without us
```

```
ing any element
dp[i][0] = true empty subset has zero sum so we can create su
bset of zero sum by using any number of elements
i = 1 to nums.length
j = 1 to totalSum
dp[i][j] =
  dp[i-1][j] || dp[i-1][j-nums[i-1]] if j >= nums[i-1]
  dp[i-1][j]  otherwise
```

LIS Longest increasing subsequence

```
Input array nums[] containing n integers
Dp state
dp[i] represents longest increasing subsequence till i index
where nums[i] is the largest element or nums[i] included in i
ncreasing subsequence.init state
dp[i] = 1
i = 1 to nums.length-1
j = 0 to i-1
dp[i] = max(dp[i], 1 + dp[j])   if nums[j] < nums[i]
        dp[i]    otherwise
```

2 Player Game

Given n coins two players A and B are playing a game with alternate turns. In each turn player can either choose first coin or last coin. Find maximum sum of coins player A can get.

```
Input array coins[] containing n positive integers
Dp state
dp[i][j] represents maximum coin sum player A can get if he m
ake first turn.dp[i][j] = coins[i] if i == j
dp[i][j] = MAX(coins[i], coins[j]) if j=i+1len = 3 to arr.len
gth
i = 0 to arr.length - len
```

```
j = i+len-1dp[i][j] = MAX(coins[i] + MIN(dp[i+2][j],dp[i+1][j
-1]), coins[j] + MIN(dp[i+1][j-1], d[i][j-2]))
```

Counting paths in matrix.

Given a nxm matrix, count number of ways to reach from top left corner to bottom right corner.

```
Input 2 integers n and m
Dp state
dp[i][j] represents no of ways to reach from cell(0,0) to cel
l(i,j)
dp[0][0] = 1i = 0 to n-1
j = 0 to n-1dp[i][j] = dp[i-1][j] + dp[i][j-1]
```