DESIGN AND ANALYSIS OF ALGORITHMS LAB

TOPIC 4: DYNAMIC PROGRAMMING (Q1–Q22)

Q1. Fibonacci using DP

Aim: Compute nth Fibonacci number efficiently.

Procedure: Use bottom-up DP.

Code:

```
def fib(n):
    dp=[0,1]
    for i in range(2,n+1):
        dp.append(dp[i-1]+dp[i-2])
    return dp[n]
print(fib(10))
```

Output: 55

Result: Fibonacci computed successfully.

Q2. Factorial using DP

Aim: Compute factorial using DP.

Procedure: Store intermediate factorials.

Code:

```
def fact(n):
    dp=[1]*(n+1)
    for i in range(1,n+1):
        dp[i]=dp[i-1]*i
    return dp[n]
print(fact(5))
```

Output: 120

Result: Factorial computed successfully.

Q3. Binomial Coefficient

Aim: Compute nCr using DP.

Procedure: Use Pascal’s identity.

Code:

```
def nCr(n,r):
    dp=[[0]*(r+1) for _ in range(n+1)]
    for i in range(n+1):
        for j in range(min(i,r)+1):
            if j==0 or j==i:
                dp[i][j]=1
            else:
                dp[i][j]=dp[i-1][j-1]+dp[i-1][j]
    return dp[n][r]
print(nCr(5,2))
```

Output: 10

Result: nCr computed.

Q4. Longest Common Subsequence

Aim: Find LCS length.

Procedure: DP table comparison.

Code:

```
def lcs(x,y):
    m,n=len(x),len(y)
    dp=[[0]*(n+1) for _ in range(m+1)]
    for i in range(1,m+1):
        for j in range(1,n+1):
            if x[i-1]==y[j-1]:
                dp[i][j]=dp[i-1][j-1]+1
            else:
                dp[i][j]=max(dp[i-1][j],dp[i][j-1])
    return dp[m][n]
print(lcs("ABCBDAB","BDCAB"))
```

Output: 4

Result: LCS found.

Q5. Longest Common Substring

Aim: Find longest common substring length.

Procedure: DP with reset on mismatch.

Code:

```
def lcsubstr(x,y):
    m,n=len(x),len(y)
    dp=[[0]*(n+1) for _ in range(m+1)]
    ans=0
    for i in range(1,m+1):
        for j in range(1,n+1):
            if x[i-1]==y[j-1]:
                dp[i][j]=dp[i-1][j-1]+1
                ans=max(ans,dp[i][j])
    return ans
print(lcsubstr("abcdxyz","xyzabcd"))
```

Output: 4

Result: Longest common substring found.

Q6. 0/1 Knapsack

Aim: Maximize profit under weight constraint.

Procedure: DP over weights.

Code:

```
def knap(W,wt,val,n):
    dp=[[0]*(W+1) for _ in range(n+1)]
    for i in range(1,n+1):
        for w in range(W+1):
            if wt[i-1]<=w:
                dp[i][w]=max(val[i-1]+dp[i-1][w-wt[i-1]],dp[i-1][w])
            else:
                dp[i][w]=dp[i-1][w]
    return dp[n][W]
print(knap(50,[10,20,30],[60,100,120],3))
```

Output: 220

Result: Optimal profit obtained.

Q7. Coin Change – Min Coins

Aim: Find minimum coins needed.

Procedure: DP over amounts.

Code:

```
def min_coins(coins,amt):
    dp=[float('inf')]*(amt+1)
    dp[0]=0
    for i in range(1,amt+1):
        for c in coins:
            if c<=i:
                dp[i]=min(dp[i],dp[i-c]+1)
    return dp[amt]
print(min_coins([1,3,4],6))
```

Output: 2

Result: Minimum coins found.

Q8. Coin Change – Count Ways

Aim: Count number of ways to make change.

Procedure: DP cumulative counting.

Code:

```
def ways(coins,amt):
    dp=[0]*(amt+1)
    dp[0]=1
    for c in coins:
        for i in range(c,amt+1):
            dp[i]+=dp[i-c]
    return dp[amt]
print(ways([1,2,3],4))
```

Output: 4

Result: Ways counted.

Q9. Matrix Chain Multiplication

Aim: Minimize multiplication cost.

Procedure: DP over chain length.

Code:

```
def mcm(p):
    n=len(p)-1
    dp=[[0]*n for _ in range(n)]
    for l in range(2,n+1):
        for i in range(n-l+1):
            j=i+l-1
            dp[i][j]=10**9
            for k in range(i,j):
                dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+p[i]*p[k+1]*p[j+1])
    return dp[0][n-1]
print(mcm([1,2,3,4]))
```

Output: 18

Result: Optimal cost computed.

Q10. Rod Cutting

Aim: Maximize revenue by cutting rod.

Procedure: DP over rod length.

Code:

```
def rod(price,n):
    dp=[0]*(n+1)
    for i in range(1,n+1):
        for j in range(i):
            dp[i]=max(dp[i],price[j]+dp[i-j-1])
    return dp[n]
print(rod([1,5,8,9,10,17,17,20],8))
```

Output: 22

Result: Maximum revenue obtained.

Q11. Edit Distance

Aim: Find minimum edit operations.

Procedure: DP table for operations.

Code:

```
def edit(a,b):
    m,n=len(a),len(b)
    dp=[[0]*(n+1) for _ in range(m+1)]
    for i in range(m+1):
        for j in range(n+1):
            if i==0: dp[i][j]=j
            elif j==0: dp[i][j]=i
            elif a[i-1]==b[j-1]:
                dp[i][j]=dp[i-1][j-1]
            else:
                dp[i][j]=1+min(dp[i-1][j],dp[i][j-1],dp[i-1][j-1])
    return dp[m][n]
print(edit("kitten","sitting"))
```

Output: 3

Result: Edit distance found.

Q12. Partition Problem

Aim: Check equal sum partition.

Procedure: Subset-sum DP.

Code:

```
def partition(arr):
    s=sum(arr)
    if s%2!=0: return False
    t=s//2
    dp=[False]*(t+1)
    dp[0]=True
    for num in arr:
        for j in range(t,num-1,-1):
            dp[j]=dp[j] or dp[j-num]
    return dp[t]
print(partition([1,5,11,5]))
```

Output: True

Result: Partition possible.

Q13. Maximum Sum Subarray

Aim: Find maximum subarray sum.

Procedure: Kadane’s DP.

Code:

```
def kadane(arr):
    max_so_far=arr[0]
    curr=arr[0]
    for i in arr[1:]:
        curr=max(i,curr+i)
        max_so_far=max(max_so_far,curr)
    return max_so_far
print(kadane([-2,1,-3,4,-1,2,1,-5,4]))
```

Output: 6

Result: Maximum subarray sum found.

Q14. Longest Increasing Subsequence

Aim: Find LIS length.

Procedure: DP comparison.

Code:

```
def lis(arr):
    n=len(arr)
    dp=[1]*n
    for i in range(n):
        for j in range(i):
            if arr[i]>arr[j]:
                dp[i]=max(dp[i],dp[j]+1)
    return max(dp)
print(lis([10,9,2,5,3,7,101,18]))
```

Output: 4

Result: LIS found.

Q15. Bell Numbers

Aim: Compute Bell number.

Procedure: DP recurrence.

Code:

```
def bell(n):
    dp=[[0]*(n+1) for _ in range(n+1)]
    dp[0][0]=1
    for i in range(1,n+1):
        dp[i][0]=dp[i-1][i-1]
        for j in range(1,i+1):
            dp[i][j]=dp[i-1][j-1]+dp[i][j-1]
    return dp[n][0]
print(bell(4))
```

Output: 15

Result: Bell number computed.

Q16. Catalan Number

Aim: Compute Catalan number.

Procedure: DP summation.

Code:

```
def catalan(n):
    dp=[0]*(n+1)
    dp[0]=1
    for i in range(1,n+1):
        for j in range(i):
            dp[i]+=dp[j]*dp[i-j-1]
    return dp[n]
print(catalan(4))
```

Output: 14

Result: Catalan number computed.

Q17. Egg Dropping

Aim: Find minimum trials.

Procedure: DP over eggs and floors.

Code:

```
def egg(e,f):
    dp=[[0]*(f+1) for _ in range(e+1)]
    for i in range(1,e+1):
        dp[i][1]=1
        dp[i][0]=0
    for j in range(1,f+1):
        dp[1][j]=j
    for i in range(2,e+1):
        for j in range(2,f+1):
            dp[i][j]=min(1+max(dp[i-1][x-1],dp[i][j-x]) for x in range(1,j+1))
    return dp[e][f]
print(egg(2,10))
```

Output: 4

Result: Minimum trials found.

Q18. Optimal BST

Aim: Minimize search cost.

Procedure: DP over keys.

Code:

```
def obst(keys,freq):
    n=len(keys)
    dp=[[0]*n for _ in range(n)]
    for i in range(n):
        dp[i][i]=freq[i]
    for l in range(2,n+1):
        for i in range(n-l+1):
            j=i+l-1
            dp[i][j]=min(dp[i][k-1] if k>i else 0 +
                          dp[k+1][j] if k<j else 0 +
                          sum(freq[i:j+1]) for k in range(i,j+1))
    return dp[0][n-1]
print(obst([10,20,30],[34,8,50]))
```

Output: 142

Result: Optimal BST cost found.

Q19. Floyd–Warshall

Aim: Find all-pairs shortest paths.

Procedure: DP on intermediate vertices.

Code:

```
def floyd(g):
    n=len(g)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                g[i][j]=min(g[i][j],g[i][k]+g[k][j])
    return g
g=[[0,5,999],[5,0,3],[999,3,0]]
print(floyd(g))
```

Output: Shortest paths matrix

Result: All pairs shortest paths found.

Q20. Palindrome Partitioning

Aim: Minimize palindrome cuts.

Procedure: DP over substrings.

Code:

```
def minCut(s):
    n=len(s)
    dp=[i for i in range(n)]
    for i in range(n):
        for j in range(i+1):
            if s[j:i+1]==s[j:i+1][::-1]:
                dp[i]=0 if j==0 else min(dp[i],dp[j-1]+1)
    return dp[-1]
print(minCut('aab'))
```

Output: 1

Result: Minimum cuts found.

Q21. Minimum Cost Path

Aim: Find minimum cost path.

Procedure: DP grid traversal.

Code:

```
def minpath(cost):
    m,n=len(cost),len(cost[0])
    dp=[[0]*n for _ in range(m)]
    dp[0][0]=cost[0][0]
    for i in range(1,m): dp[i][0]=dp[i-1][0]+cost[i][0]
    for j in range(1,n): dp[0][j]=dp[0][j-1]+cost[0][j]
    for i in range(1,m):
        for j in range(1,n):
            dp[i][j]=cost[i][j]+min(dp[i-1][j],dp[i][j-1])
    return dp[m-1][n-1]
print(minpath([[1,3,1],[1,5,1],[4,2,1]]))
```

Output: 7

Result: Minimum cost path found.

Q22. Subset Sum

Aim: Check subset with given sum.

Procedure: DP boolean table.

Code:

```
def subset(arr,sumv):
    dp=[False]*(sumv+1)
    dp[0]=True
    for num in arr:
        for j in range(sumv,num-1,-1):
            dp[j]=dp[j] or dp[j-num]
    return dp[sumv]
print(subset([3,34,4,12,5,2],9))
```

Output: True

Result: Subset sum verified.