DP Concepts

1.DP

it is an enhanced **recursion**

how to detect - 1. choice 2. optimal- min,max,largest

DP →recursion soln→memoization→topdown

questions -

1. 0- 1 knapsack - subset sum,equal sum,count of subset, min subset,target sum,#offset
2. Unbounded knapsack
3. fibonacci
4. lcs
5. LI
6. Kadanes algo
7. Dp on trees
8. matrix chain multiplication
9. DP on grid
10. others

0 - 1 knapsack

DP →recursion soln→memooization→topdown

choice diagram

wt[] : 1,3,4,5

val[]: 1,4,5,7

W: 7

recursive find : fun(ip) → fun (ip—)

base cond → think of the smallest valid ip

if(n==0||w==0) return 0 —Base cond

if(wt[n-1]≤w){

val[n-1] + knapsack(wt,val,W-wt[n-1],n-1))

}

if(wt[n-1] > W{

return knapsack(wt,val,W,n-1);

}

Function

```
int knapsack(int wt[], int val[], int W, int n)
{
    if (n == 0 || W == 0)
    {
        return 0;
    }
    if (wt[n - 1]≤w)
    {

return max(val[n-1] + knapsack(wt,val,W-wt[n-1],n-1),knapsack(wt
    }

    else if (wt>W)
    {
       knapsack(wt,val,W,n-1)
    }
```

```
}
```

Memoization

W and n is changing so make n by W matrix

int t [n+1][W+1]

initialise all matrix by -1 → memset(t,-1,sizeof(t))

CODE

```
#include <bits/stdc++.h>
using namespace std;

int t[102][1002]; // global memoization table

int knapsack(int wt[], int val[], int W, int n)
{
    if (n == 0 || W == 0)
    {
        return 0;
    }

    if (t[n][W] != -1)
    {
        return t[n][W];
    }

    if (wt[n - 1] <= W)
    {
        return t[n][W] = max(val[n - 1] + knapsack(wt, val, W -
    }
    else if (wt[n - 1] > W)
    {
```

```
        return t[n][W] = knapsack(wt, val, W, n - 1);
    }
}

int main()
{
    int n, W;

    cout << "Enter the number of items: ";
    cin >> n;

    int wt[n], val[n];

    cout << "Enter the weights of the items: ";
    for (int i = 0; i < n; i++)
    {
        cin >> wt[i];
    }

    cout << "Enter the values of the items: ";
    for (int i = 0; i < n; i++)
    {
        cin >> val[i];
    }

    cout << "Enter the maximum capacity of the knapsack: ";
    cin >> W;

    memset(t, -1, sizeof(t)); // initialize memoization table wi

    int result = knapsack(wt, val, W, n);
    cout << "The maximum value that can be obtained is: " << res

    return 0;
```

```
}
```

TOP Down

if(n==0||w==0) return 0 —Base cond in recursive

for (int i = 0; i < n+1; i++) // for top down
{
  for (int j = 0; j < W+1; j++)
  {
    if(i==0 || j ==0){
      t[i][j] = 0;
    }
  }

}

```
if (wt[n - 1] <= W) // in recursive
    {
        return t[n][W] = max(val[n - 1] + knapsack(wt, val, W -
    }

    else if (wt[n - 1] > W)
    {
        return t[n][W] = knapsack(wt, val, W, n - 1);
    }
```

```
if (wt[n - 1] <= W) // in TOP Down
    {
    t[n][w] = max(val[n-1] + t[W-wt[n-1]][n-1]) , t[n-1][W]
      }

       else if (wt[n - 1] > W)
    {
```

```
          t[n][W] = t[n-1][W];
     }
```

1.Subset Sum Problem

t[n+1][w+1] → in knapsack

t[n+1][sum+1] → in subset sum

for(int i){
for(int j){

if(i==0) t[i][j] = false

if(j==0) t[i][j] = true

}}

```
if (wt[i - 1] <= j) // in TOP Down
    {
    t[n][w] = max(val[i-1] + t[W-wt[i-1]][n-1]) , t[i-1][j]
      }

       else if (wt[i - 1] > j)
    {
        t[i][j] = t[i-1][j];
    }
```

```
if (arr[i - 1] <= j) // in subset
    {
    t[n][w] = t[i][j-arr[i-1]]) || t[i-1][j]
      }

       else if (wt[i - 1] > j)
    {
```

```
            t[i][j] = t[i-1][j];
        }
```

1. Partition Equal Subset Sum leetcode

```
#include <numeric>
using namespace std;

class Solution {
public:
    bool canPartition(vector<int>& nums) {
        int sum = 0;
        for(int i = 0;i<nums.size();i++){
            sum+=nums[i];
        }

        // If the sum is odd, we can't partition it into two equ
        if (sum % 2 != 0) return false;

        int target = sum / 2;
        return isSubsetSum(nums, target);
    }

    bool isSubsetSum(vector<int>& nums, int target) {
        int n = nums.size();

        // Dynamic allocation of the DP table
        vector<vector<bool>> t(n + 1, vector<bool>(target + 1, f

        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= target; j++) {
                if (i == 0) {
```

```
                    t[i][j] = false;
                } else if (j == 0) {
                    t[i][j] = true;
                } else {
                    t[i][j] = false;
                }
            }
        }

        // Fill the dp array
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= target; j++) {
                if (nums[i - 1] <= j) {
                    t[i][j] = t[i - 1][j] || t[i - 1][j - nums[
                } else {
                    t[i][j] = t[i - 1][j];
                }
            }
        }

        return t[n][target];
    }
};
```

3. Count of subset of a given sum

```
if (arr[i - 1] <= j) // in subset
    {
    t[n][w] = t[i][j-arr[i-1]]) || t[i-1][j]
      }
```

```
       else if (wt[i - 1] > j)
    {
        t[i][j] = t[i-1][j];
    }
```

```
if (arr[i - 1] <= j) // in this || -> +
    {
    t[n][w] = t[i][j-arr[i-1]]) + t[i-1][j]
      }

       else if (wt[i - 1] > j)
    {
        t[i][j] = t[i-1][j];
    }

    return t[n][w] // return int
```