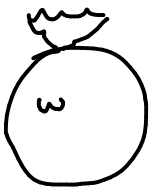
CLASS 81

10 May 2023 21:10

Class 81

Dynamic Programming

0 - 1 Knapsack



item1, item2, item3 - - item n

w1 w2 - - - wn

v1 v2 - - - vn

subset of items selected

Σ w(item i) ≤ W

Unbounded Knapsack — ∞ supply of each item

valu = 5, 11, 13

weigw = 2, 4, 6

n = 10^4, W = 10^5 → 10^9

6, 4 = 24

6, 2, 2 = 23

4, 4, 2 = 27

2, 2, 2, 2, 2 = 20

```
vector<vector<int>> dp(n+1, vector<int>(w+1, -1))
int knap( vector<int> &wt, vector<int> & valu, int idx, int w)
{
    if (idx == 0 or w == 0)
        return 0;
    if(dp[idx][w] != -1) return dp[idx][w];

    int take = 0; int discard = 0;

    if (wt[idx-1] <= w)
    { take = knap( wt, valu, idx, w - wt[idx-1]) + value[idx-1];
    }
    discard = knap (wt, valu, idx-1, w);
    dp[idx][w] =
    return max (take, discard);
```

n = 5

1, 2, 3, 4, 5

0 1 2 3 4

idx = 4

idx = 5

1-2

2 D DP

Memoize

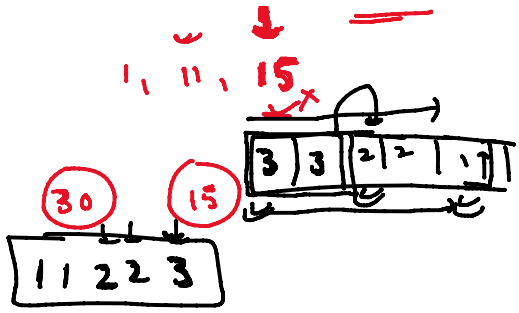
1, 11, 15

dp[idx][w] =

return max (take , discard) ;

}

DP → 2D 1DP



Tabulation

Unbounded knapsack → using 2D DP

→ Into 1D DP

Max Profit ?

O time

n = 10^4 w = 10^5

vector < int > dp(w+1, -1)

O time

int knap(vector <int> &wt, vector <int> &value, int w)

Intuition

Unbounded

if (w == 0)

return 0 ;

1D DP

if (dp[w] != -1) return dp[w]

int ans = 0 ;

for(int i = 0; i < wt.size(); i++)

{ if (wt[i] ≤ w)

{ ans = max(ans, knap(wt, value, w - wt[i]) + value[i]);

} }

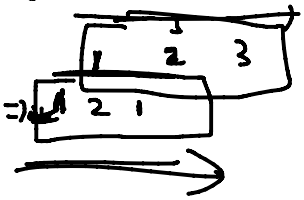
dp[w] =

return ans ;

}

n

T.C = O(n*w)



Total steps = w * n = n * w

Tabulation

Vector <int> dp(w+1, -1) ;

dp[idx] ans

dp[0] =

su-100 = Subset → 10 → 10

1000   700-800   10   10   10   100

```
int unboundedKnapsack(int n, int w, vector<int> &profit, vector<int> &weight)
{
    // Write Your Code Here.
    int i,j,k;
    vector<int>dp(w+1,0);
    dp[0]=0;
    for(i=1;i<=w;i++)
    {
        for(j=0;j<n;j++){
            if(weight[j]<=i){
                dp[i]=max(dp[i],dp[i-weight[j]]+profit[j]);
            }
        }
    }
    return dp[w];
}
```

1DDP

dp[ ]

0 1 2 3 4 ... w

Memoization + Tabulation → using 2D and 1D P

[Target sum, knapsack, subset sum,]

indendecision

Subset sum equals K

4 3 2 1   sum = 5

(3 2) (4 1)

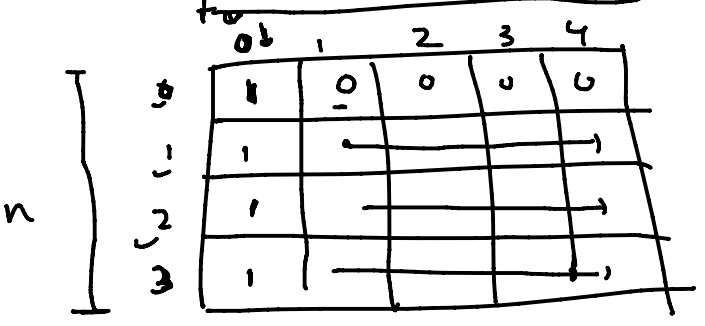
Subset

Combination sum I

```
bool subset( int idx, int sum, vector<int>& nums )
{
    if( sum == 0 )
        return true;
    if( idx == 0 )
        return false;
    if( dp[idx][sum] != -1 ) return dp[idx][sum];
    int take = 0; int discard = 0;
    if( nums[idx-1] <= sum )
    {
        take = subset( idx-1, sum - nums[idx-1], nums );
    }
    discard = subset( idx-1, sum, nums );
    return take or discard;
    dp[idx][sum] =
}
```

Tabulation

```
vector<vector<int>> dp( n+1, vector<int>( S+1, -1 ) );
```

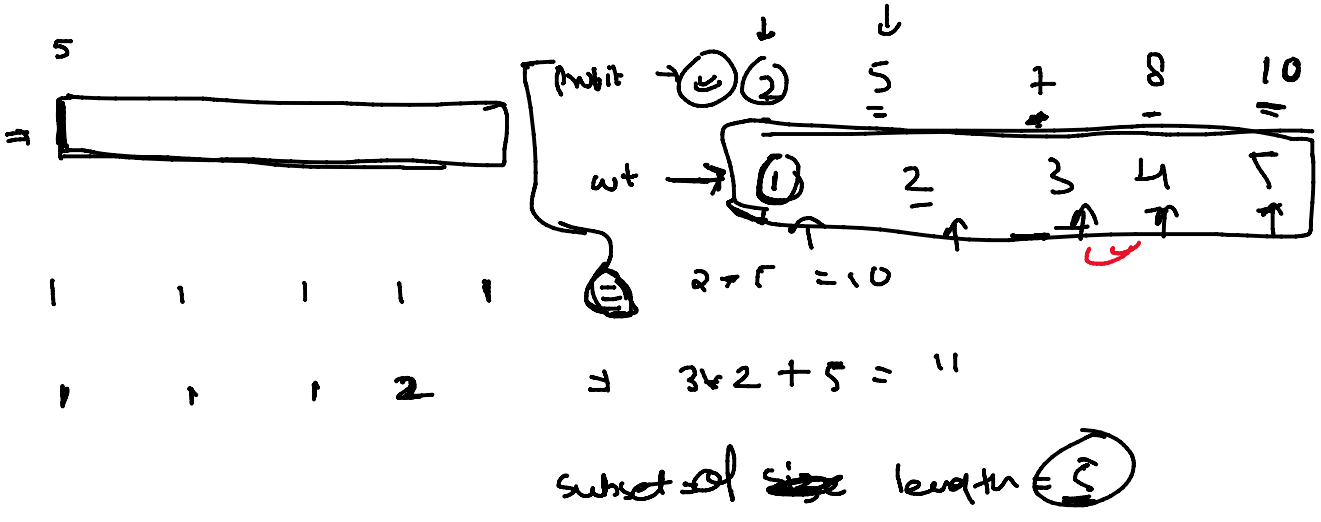



```
For( i=0; i<=n; i++ )
{ dp[i][0] = 1;
}

For( i=1; i<=S; i++ )
{ dp[0][i] = 0;
}

For( i=1; i<=n; i++ )
{ For( j=1; j<=S; j++ )
  { int one = 0;
    if( ..... [i-1] <= j )
```

```
int ans = 0;
if ( num[i-1] <= j )
{   ans = ans or dp[i-1][j - num[i-1]];
}
ans = ans or dp[i-1][j];
dp[i][j] = ans;
}
}
return dp[n][sum]
```

Tabulation

```
bool subsetSumToK(int n, int k, vector<int> &arr) {
    // Write your code here.
    vector<vector<int>>dp(k+1,vector<int>(n+1,0));
    for(int i=0;i<=n;i++){
        dp[0][i]=1;
    }
    int i,j;
    for(i=1;i<=k;i++){
        dp[i][0]=0;
    }
    for(i=1;i<=k;i++){
        for(j=1;j<=n;j++){
            if(arr[j-1]<=i){
                dp[i][j]=dp[i-arr[j-1]][j-1] or dp[i][j-1];
            }
            else
            {
                dp[i][j]=dp[i][j-1];
            }
        }
    }
    return dp[k][n];
}
```



2D, 1D DP

Memoization

Tabulation

Max Profit

```
int rodCut(vector<int>&price,int length,int peice,vector<vector<int>>&dp){
    if(length==0 or peice==0)
    return 0;
    if(dp[length][peice]!=-1)
    return dp[length][peice];
    int op1=0;
    int op2=0;
    if(peice<=length){
        op1=rodCut(price,length-peice,peice,dp)+price[peice-1];
    }
  op2=rodCut(price,length,peice-1,dp);
    return dp[length][peice]=max(op1,op2);
}
```

5 → 3

2DDP

```
int rodCut(vector<int>&price,int length,vector<int>&dp){
    if(length==0)
    return 0;
    if(dp[length]!=-1)
    return dp[length];
    int ans=0;
    for(int i=1;i<=price.size();i++){
        if(length-i>=0){
            ans=max(ans,rodCut(price,length-i,dp)+price[i-1]);
        }
    }
    return dp[length]=ans;
}
```

1D DP

```
vector<int>dp(n+1,0);

for(i=0;i<=n;i++){
    for(j=1;j<=m;j++){
        if(i==0){
            dp[i]=0;
        }
        else
        {
            if(j<=i){
                dp[i]=max(dp[i],dp[i-j]+price[j-1]);
            }
        }
    }
}
return dp[n];
```

1DDP Tabulation

(2) Unbounded Knapsack, subset sum, Rod cutting problem, Coin change, Fibonacci, Frog jump (2)