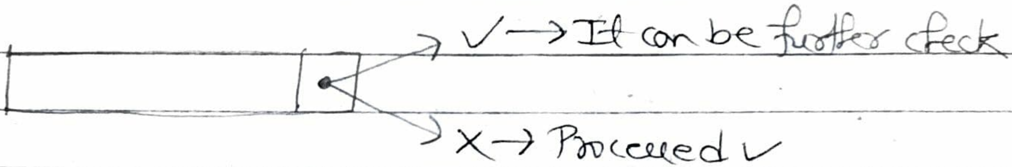
## * Unbounded Knapsack- (multiple occurrences)

Flow-

→ Recall differences
→ See code different variation.

Related problems-

→ Rod Cutting
→ Coin change I
→ Coin change II
→ Maximum Ribbon cut



Multiple occurrences- means we can use items as many times we want. But if we don't choose that item then we will never come back to that item.

Program-

→ weight

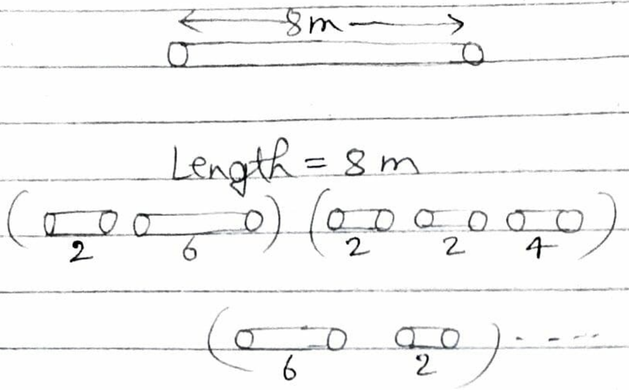
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | | | | | | |
| 2 | 0 | | | | | | |
| 3 | 0 | | | | | | |
| 4 | 0 | | | | | | |

↓ size

```
if (wt[i-1] <= j)
    t[i][j] = max (val[i-1] + t[i][j-wt[i-1]]
                             , t[i-1][j]);

else
    t[i][j] = t[i-1][j]
```

# * Rod Cutting Problem-



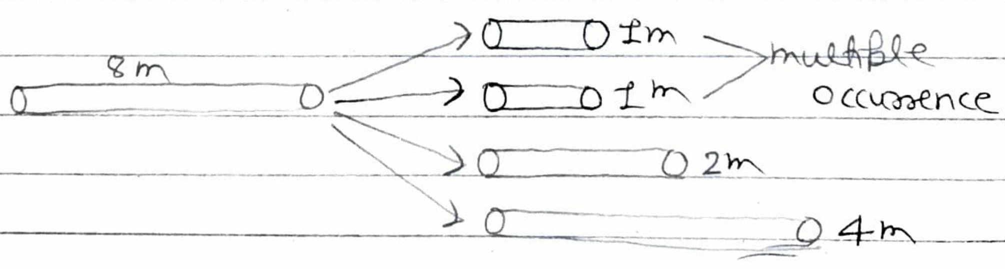
Flow ↓

- → Problem statement
- → Matching
- → How to identify
  - 0-1
  - unbounded
- → Code variation

Given a rod of length n inches (or n meter) & an array of prices that includes prices of all pieces of size smaller than n. Determine the maximum value obtainable by cutting up the rod & selling the pieces.

| Length[] : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price [] : | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Length of rod : 8 m



t[n+1][Length+1]

Program-

```
if ( Length[i-1] <= j)
    t[i][j] = max (Price[i-1] + t[i][j-
                  Length[i-1]], t[i-1][j]);
else
    t[i][j] = t[i-1][j];
```

## * Coin change problem - Maximum no of ways:-

```
coin[] : | 1 | 2 | 3 |  ---> unlimited coins
sum    :  5
```

The coin change problem deals with finding the total no of ways that an amount of money can be made using specific coins only.

Ex- for, coins[] = 1,2,3
sum = 5

then,
2+3 = 5
1+2+2 = 5
1+1+3 = 5
1+1+1+1+1 = 5
1+1+1+2 = 5

} 5 ways

multiple occurences

It is same as "count of subset sum with given sum"
P.no - 10

t[n+1][sum+1]

```
if (coin[i-1] <= j)
    t[i][j] = t[i][j-coin[i-1]] + t[i-1][j];
else
    t[i][j] = t[i-1][j];
```

## * Coin change problem – Minimum no of coins –

Coins[] : | 1 | 2 | 3 | → unlimited coins

Sum : 5

| | No of coins |
|---|---|
| 2+3 = 5 | → 2 → min no of coin ✓ |
| 1+2+2 = 5 | → 3 |
| 1+1+1+2 = 5 | → 4 |
| 1+1+3 = 5 | → 3 |
| 1+1+1+1+1 = 5 | → 5 |

INT_MAX-1

Initialization – t[n+1][sum+1] = t[4][6]

→ sum

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | " | ▨ | " | " | " | " |
| 1 | 0 | | | | | |
| 2 | 0 | | | | | |
| 3 | 0 | | | | | |

size(n)

> INT_MAX-1

↓

∞

We have to initialize also the 2nd row

For 2nd row initialization –

```
for (int j = 1; j <= sum+1; j++)
    if (j % arr[0] == 0)
        t[i][j] = j/arr[0];
    else
        t[i][j] = INT_MAX - 1;
```

Code-

```
for (int i=2; i<n+1; i++)
 { for (int j=1; j<sum+1; j++)
   {
   if(coin[i-1]<=j)
     t[i][j] = min(t[i][j-coin[i-1]]+1,
                   t[i-1][j])
   else
     t[i][j] = t[i-1][j]
   }
  }
```