## 4.6.1 Algorithm

```
Algorithm Dynamic_Knapsack(n,W,w[ ],v[ ])
//Problem Description:This algorithm is for obtaining knapsack
//solution using dynamic programming
//Input: n is total number of items, W is the capacity of
//knapsack, w[ ] stores weights of each item and v[ ] stores
//the values of each item.
//Output: Returns the total value of selected items for the
//knapsack.
for (i←0 to n) do
{
  for (j←0 to W) do
  {
    table[i,0]=0   // table initialization
    table[0,j]=0
  }
}
for (i←0 to n) do
{
  for (j←0 to W) do
  {
    if(j<w[i]) then
      table[i,j]← table[i-1,j]
    else if(j>=w[i]) then
      table[i,j]← max (table[i-1,j],(v[i]+table[i-1,j-w[i]]) )
  }
}
return table[n,W]
```

**Algorithm**

```
Algorithm Bellman Ford (vertices, edges, source)
{
// Problem Description : This algorithm finds
// the shortest path using Bellman Ford method
for (each vertex v)
{
        if (v is source) then
        v distance ← 0
        else
        v. distance ← infinity
        v. prede ← Null
}
for (i←1 to toal_vertices – 1)
{
        for (each edge uv)
        {
```

Graph initialization

```
        U ← uv. source                                          Newly obtained
        V ← uv. desination                                      minimum distance
        if (v. distance > u. distance + uv. weight )    then
        {
                v. distance ← u. distance + uv. weight
                v. prede ← u
        }
    }                                                           Relaxing edges
    for (each edge uv)
    {
        u ← uv. source
        v ← uv. destination
        if (v. distance > u.distance + uv. weight) then
        {
                Write ("Graph has negative edges")
                return False
        }
    }
} // end of for return True
} // end of algorithm
```

In above algorithm, we have used a term "relaxing edges". The process of relaxing