# WEEK 6

Y. Shamil Ahamed
1BM21CS248.

## 1/0 Knapsack Problem using Dynamic Program:

## INPUT:

```
#include<stdio.h>

int max(int a, int b){
    return (a > b)? a : b;
}

int knapSack(int W, int wt[], int val[], int n)
{
   int i, j;
   int K[n+1][W+1];
    for (i=0;i<=n;i++){
        for (j=0;j<=W;j++){
            if (i==0 || j==0)
                K[i][j] = 0;
            else if (wt[i-1] <= j)
                    K[i][j] = max(val[i-1]+K[i-1][j-wt[i-1]],K[i-
1][j]);
            else
                    K[i][j]=K[i-1][j];
        }
    }
    return K[n][W];
}

int main()
{
    int i, n, prf[20], wt[20], W;
```

```
    printf("Enter number of items:\n");
    scanf("%d", &n);

    for(i = 0;i < n; ++i){
        printf("Enter weight and profit of item %d:\n",i+1);
        scanf("%d%d",&wt[i],&prf[i]);
    }

    printf("Enter size of knapsack:\n");
    scanf("%d", &W);
    int final = knapSack(W,wt,prf,n);
    printf("The final value is:\n%d", final);
    return 0;
}
```

## OUTPUT:

```
Enter number of items:
4
Enter weight and profit of item 1:
2 12
Enter weight and profit of item 2:
1 10
Enter weight and profit of item 3:
3 20
Enter weight and profit of item 4:
2 15
Enter size of knapsack:
5
The final value is:
37
```

# Floyd's Algorithm [represents shortest path]:

## INPUT:

```
#include<stdio.h>

int min(int a,int b) {
    if(a<b) return(a);
    else return(b);
}

void floyds(int p[10][10],int n) {
    int i,j,k;
    for (k=1;k<=n;k++)
    {
        for (i=1;i<=n;i++)
        {
            for (j=1;j<=n;j++)
                    p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
        }
    }
}

int main() {
    int p[10][10],w,n,e,u,v,i,j;
    printf("\n Enter the number of vertices and edges:");
    scanf("%d %d",&n,&e);
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
        {
            if(i==j)
                p[i][j]=0;
            else
                p[i][j]=999;
        }
    }
    for (i=1;i<=e;i++) {
        printf("\nEnter the end vertices of edge %d with its
weight:\n",i);
        scanf("%d %d %d",&u,&v,&w);
        p[u][v]=w;
    }
```

```
    printf("\n Matrix of input data:\n");
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
            printf("%d\t\t",p[i][j]);
        printf("\n");
    }
    floyds(p,n);
    printf("\n Final Matrix:\n");
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
            printf("%d \t",p[i][j]);
        printf("\n");
    }
    return 0;
}
```

## OUTPUT:

```
Enter the number of vertices and edges:4 5
Enter the end vertices of edge 1 with its weight:
2 1 2
Enter the end vertices of edge 2 with its weight:
1 3 3
Enter the end vertices of edge 3 with its weight:
3 4 1
Enter the end vertices of edge 4 with its weight:
3 2 7
Enter the end vertices of edge 5 with its weight:
4 1 6
Matrix of input data:
0       999     3       999
2       0       999     999
999     7       0       1
6       999     999     0

 Final Matrix:
0   10  3   4
2   0   5   6
7   7   0   1
6   16  9   0
```