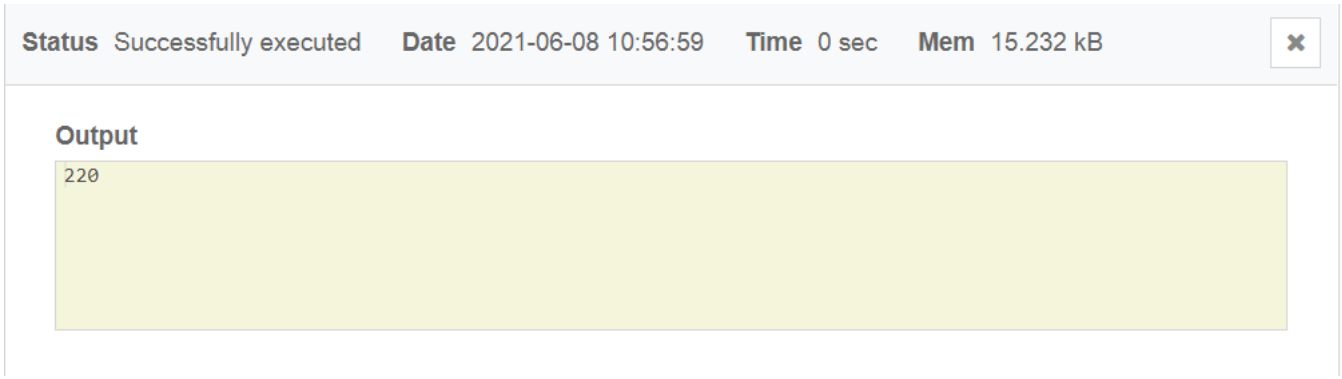
LAB Assignment 4	
NAME: **AVIL HARSH**	Roll NUMBER: **IMH/10064/18**

Assignment 1	Implement the 0/1 Knapsack problem using (a) Dynamic Programming method (b) Greedy method.	Date: 08/06/21

Solution (a) Dynamic Programming method

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

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

int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n + 1][W + 1];

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

int main()
{
   int val[] = { 60, 100, 120 };
   int wt[] = { 10, 20, 30 };
   int W = 50;
   int n = sizeof(val) / sizeof(val[0]);

   cout << knapSack(W, wt, val, n);

   return 0;
}
```



Solution (b) Greedy method

In this item cannot be broken which means thief should take the item as a whole or should leave it. That's why it is called **0/1 knapsack Problem**.

i) Each item is taken or not taken.

ii) Cannot take a fractional amount of an item taken or take an item more than once.

iii) It cannot be solved by the Greedy Approach because it is enable to fill the knapsack to . . capacity.

iv) **Greedy Approach** doesn't ensure an Optimal Solution.

Assignment 2	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	Date: 08/06/21

```
#include <limits.h>
#include <stdio.h>

#define V 9

int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}

void printSolution(int dist[])
{
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
   int dist[V];

   bool sptSet[V];

   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;

   dist[src] = 0;

   for (int count = 0; count < V - 1; count++)
   {
      int u = minDistance(dist, sptSet);
      sptSet[u] = true;

      for (int v = 0; v < V; v++)

         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
            && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
   }

   printSolution(dist);
}

int main()
{
```

```
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

Status Successfully executed **Date** 2021-06-08 11:12:23 **Time** 0 sec **Mem** 15.232 kB

Output

```
Vertex      Distance from Source
0        0
1        4
2        12
3        19
4        21
```

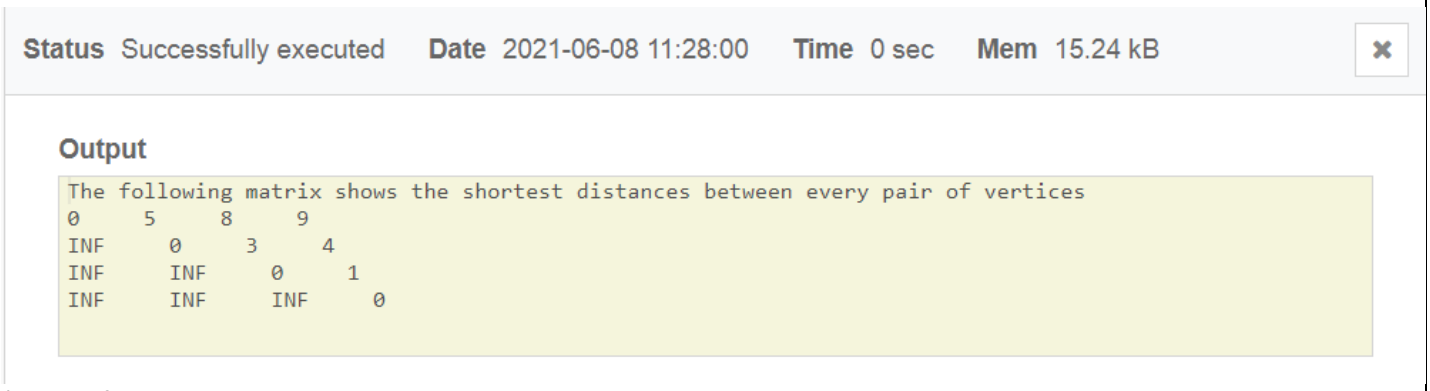
Output

```
3        19
4        21
5        11
6        9
7        8
8        14
```

Assignment 3(a)	Write C++ programs to Implement All-Pairs Shortest Paths problem using Floyd's algorithm.	Date: 08/06/21

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

#define V 4

#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall(int graph[][V])
{
   int dist[V][V], i, j, k;

   for (i = 0; i < V; i++)
      for (j = 0; j < V; j++)
         dist[i][j] = graph[i][j];

   for (k = 0; k < V; k++)
   {
      for (i = 0; i < V; i++)
      {
         for (j = 0; j < V; j++)
         {
            if (dist[i][j] > (dist[i][k] + dist[k][j])
               && (dist[k][j] != INF
                  && dist[i][k] != INF))
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }

   printSolution(dist);
}

void printSolution(int dist[][V])
{
   cout << "The following matrix shows the shortest "
         "distances"
         " between every pair of vertices \n";
   for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
         if (dist[i][j] == INF)
            cout << "INF"
                 << "     ";
         else
            cout << dist[i][j] << "     ";
      }
      cout << endl;
   }
}
```

```
int main()
{
   int graph[V][V] = { { 0, 5, INF, 10 },
                       { INF, 0, 3, INF },
                       { INF, INF, 0, 1 },
                       { INF, INF, INF, 0 } };

   floydWarshall(graph);
   return 0;
}
```



*INF - Infinity

Assignment 3(b)	Write C++ programs to Implement Travelling Sales Person problem using Dynamic programming.	Date: 08/06/21

```
#include <bits/stdc++.h>
using namespace std;
#define V 4

int travllingSalesmanProblem(int graph[][V], int s)
{
   vector<int> vertex;
   for (int i = 0; i < V; i++)
      if (i != s)
         vertex.push_back(i);

   int min_path = INT_MAX;
   do {

      int current_pathweight = 0;
      int k = s;
      for (int i = 0; i < vertex.size(); i++) {
         current_pathweight += graph[k][vertex[i]];
         k = vertex[i];
      }
      current_pathweight += graph[k][s];

      min_path = min(min_path, current_pathweight);

   } while (
      next_permutation(vertex.begin(), vertex.end()));

   return min_path;
}

int main()
{
   int graph[][V] = { { 0, 10, 15, 20 },
                      { 10, 0, 35, 25 },
                      { 15, 35, 0, 30 },
                      { 20, 25, 30, 0 } };
   int s = 0;
   cout << travllingSalesmanProblem(graph, s) << endl;
   return 0;
}
```

Status Successfully executed **Date** 2021-06-08 11:38:26 **Time** 0 sec **Mem** 15.24 kB

Output

```
80
```