Assignment 1
Fractional Knapsack

Code:

```
/*
      Name : Sahil Kothari
      Roll no. : 33243
      batch : M10
*/

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

typedef struct item{
   int value,weight;
};

bool cmp(struct item a,struct item b){
   double r1=(double)a.value / (double)a.weight;
   double r2 = (double)b.value / (double)b.weight;
   return r1 > r2;
}

int main(){


       int n,c;
       double fvalue=0;
       cout<<"Enter the value of C"<<endl;
       cin>>c;

       cout<<"how many objects do u have?"<<endl;
       cin>>n;

       item arr[n];
       for (int i = 0; i < n; i++)
       {
           cout<<"Enter the value and weight of item "<<i+1<<":"<<endl;
           cin>>arr[i].value>>arr[i].weight;
```

```
        }
        //item arr[]={{40,80},{10,10},{50,40},{30,20},{60,90}};
        sort(arr,arr+n,cmp);
        for(int i=0;i<n;i++){
            if(arr[i].weight <=c){
                c-=arr[i].weight;
                fvalue+=arr[i].value;
            }
            else{
                fvalue+=arr[i].value*((double)c/(double)arr[i].weight);
                break;
            }
        }
        cout<<"The maximum value is : ";
        cout<<fvalue<<endl;

    return 0;
}
```

Output:

```
administrator@administrator-OptiPlex-5060:~/Desktop/33243/DAA$ cd
"/home/administrator/Desktop/33243/" && g++ knapsack_greedy.cpp -o knapsack_greedy &&
"/home/administrator/Desktop/33243/"knapsack_greedy
knapsack_greedy.cpp:6:1: warning: ‘typedef’ was ignored in this declaration
 typedef struct item{
 ^~~~~~~
Enter the value of C
50
how many objects do u have?
3
Enter the value and weight of item 1:
60 10
Enter the value and weight of item 2:
100 20
Enter the value and weight of item 3:
120 30
The maximum value is : 240
```

0/1 knapsack

Code:

```
void dpMethod(Item items[], int n, int capacity){
        int miniwt = INT_MAX;
        for (int i = 0; i < n; i++{
                miniwt = min(miniwt, items[i].weight);
        }

    // creating the table for DP
    vector<vector<int>> dp(n + 1, vector<int>(capacity));
    for (int i = 0; i <= n; i++){
        for (int w = 0; w <= capacity; w++){
                if (i == 0 || w == 0){
                        dp[i][w] = 0;
                }
                else if (items[i - 1].weight <= w){
                        dp[i][w] = max(items[i - 1].value + dp[i - 1][w - items[i - 1].weight], dp[i - 1][w]);
                }
                else{
                        dp[i][w] = dp[i - 1][w];
                }
        }
    }
```

```
        cout << "i/W\t";
        for (int i = 0; i <= capacity; i += miniwt){
                cout << i << "\t";
        }
        cout << "\n";
        for (int i = 0; i <= n; i++){
                cout << i << "\t";
                for (int j = 0; j <= capacity; j += miniwt){
                        cout << dp[i][j] << "\t";
                }
                cout << "\n";
        }
        cout << "\n\nMax Profit of Knapsack Using Dynamic Approach :: " << dp[n][capacity] << "\n\n";
}
```

Output :

```
 BY USING DYNAMIC PROGRAMMING METHOD
i/W     0       10      20      30      40      50
0       0       0       0       0       0       0
1       0       60      60      60      60      60
2       0       60      100     160     160     160
3       0       60      100     160     180     220


Max Profit of Knapsack Using Dynamic Approach :: 220
PS C:\Users\sahil\CG>
```

Assignment 2

Bellman Ford

Code:

```
/*
Name : Sahil Kothari
Roll no. : 33243
batch : M10
*/

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


struct Edge {
    int src, dest, weight;
};
struct Graph {

    int V, E;


    struct Edge* edge;
};


struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}
```

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

void BellmanFord(struct Graph* graph, int src)
{
    int V = graph->V;
    int E = graph->E;
    int dist[V];


    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;


    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX
                && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }


    for (int i = 0; i < E; i++) {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX
            && dist[u] + weight < dist[v]) {
            printf("Graph contains negative weight cycle");
```

```
            return; // If negative cycle is detected, simply
                    // return
        }
    }

    printArr(dist, V);

    return;
}


int main()
{

    int V = 5; // Number of vertices in graph
    int E = 8; // Number of edges in graph
    struct Graph* graph = createGraph(V, E);


    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;


    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;


    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;


    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;


    graph->edge[4].src = 1;
```

```
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;


    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;


    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
    graph->edge[7].weight = -3;


    BellmanFord(graph, 0);

    return 0;
}
```

Output:

```
PS C:\Users\sahil\CG> cd "c:\Users\sahil\CG\" ; if ($?) { g++
knapsack.cpp -o knapsack } ; if ($?) { .\knapsack }
Vertex Distance from Source
0               0
1               -1
2               2
3               -2
4               1
PS C:\Users\sahil\CG>
```

Assignment 3
N Queens

Code:

```
/*
      Name : Sahil Kothari
      Roll no. : 33243
      batch : M10
*/
#include<bits/stdc++.h>
using namespace std;

bool check(int board[8][8],int row,int col){

   int i,j;

   for(i=0;i<col;i++){
       if(board[row][i]){

           return false;
       }
   }

   for( i=row,j=col;i>=0 && j>=0;i--,j--){
       if(board[i][j]){
           return false;
       }
   }

   for(i=row,j=col;i<8 && j>=0;i++,j--){
       if(board[i][j]){
           return false;
       }
   }

   return true;

}
```

```
bool solve(int board[8][8],int col){

   if(col==8){
       return true;
   }

   for(int i=0;i<8;i++){
       if(check(board,i,col)){
           board[i][col]=1;
           //cout<<"Entered the value 1"<<endl;

           if(solve(board,col+1))
               return true;


               board[i][col]=0;
               //cout<<"deleted the value 1 and placed 0"<<endl;

       }
   }
   return false;
}

int main(){
   // cout<<"Enter the number of queens: "<<endl;
   // int n;
   // cin>>n;
   int board[8][8];
   for(int i=0;i<8;i++){
       for(int j=0;j<8;j++){
           board[i][j]=0;
       }
   }

   solve(board,0);


       for(int i=0;i<8;i++){
           for(int j=0;j<8;j++){
```

```
                cout<<board[i][j]<<" ";
            }
            cout<<endl;
        }

    return 0;
}
```

Output :

```
administrator@administrator-OptiPlex-5060:~/Desktop/33243$ cd "/home/administrator/Desktop/33243/" && g++ n_queens.cpp -o n_queens && "/home/administrator/Desktop/33243/"n_queens
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
administrator@administrator-OptiPlex-5060:~/Desktop/33243$
```

Assignment 4
TSP

Code:

```
/*
Name : Sahil Kothari
Roll no. : 33243
batch : M10
*/

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

// implementation of traveling Salesman Problem
int travllingSalesmanProblem(int graph[][V], int s)
{
    // store all vertex apart from source vertex
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);

    // store minimum weight Hamiltonian Cycle.
    int min_path = INT_MAX;
    do {

        // store current Path weight(cost)
        int current_pathweight = 0;

        // compute current path weight
        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];

        // update minimum
        min_path = min(min_path, current_pathweight);
```

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

    return min_path;
}

// Driver Code
int main()
{
    // matrix representation of graph
    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;
}
```

Output:

```
PS D:\c++> cd "d:\c++\" ; if ($?) { g++ TSP.cpp -o TSP } ; if ($?) { .\TSP }
Minimum cost : 80
Path Taken : 0 1 3 2 0
```