Name: Vanessa D'mello
Roll No. 8863
Branch: SE Computers A (Batch A)
Experiment 10: Bellman Ford

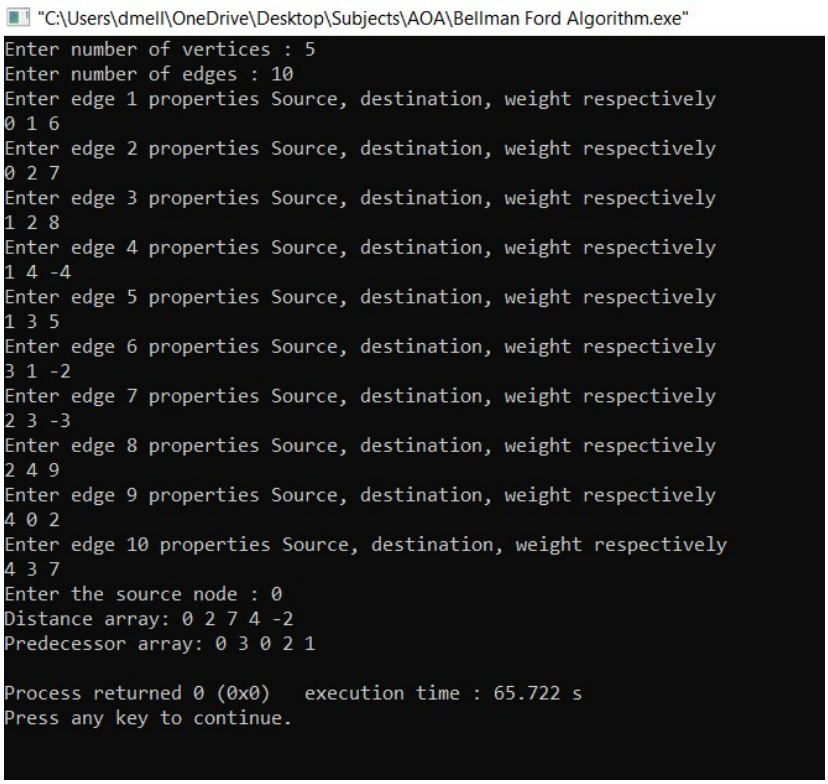
```
#include <stdio.h>
#include <stdlib.h>
#define INFINITY 99999
struct Edge {
	int u;
	int v;
	int w;
};
struct Graph {
	int V;
	int E;
	struct Edge *edge;
};
void bellmanford(struct Graph *g, int source) {
	int i, j, u, v, w;
	int tV = g->V;
	int tE = g->E;
	int d[tV];
	int p[tV];
	for (i = 0; i < tV; i++) {
		d[i] = INFINITY;
		p[i] = 0;
	}
	d[source] = 0;
	for (i = 1; i <= tV - 1; i++) {
		for (j = 0; j < tE; j++) {
			u = g->edge[j].u;
			v = g->edge[j].v;
			w = g->edge[j].w;
			if (d[u] != INFINITY && d[v] > d[u] + w) {
				d[v] = d[u] + w;
				p[v] = u;
			}
		}
	}
	// detect negative cycle
	for (i = 0; i < tE; i++) {
		u = g->edge[i].u;
		v = g->edge[i].v;
		w = g->edge[i].w;
		if (d[u] != INFINITY && d[v] > d[u] + w) {
			printf("Negative weight cycle detected!\n");
			return;
		}
	}
```

```
	printf("Distance array: ");
	display(d, tV);
	printf("Predecessor array: ");
	display(p, tV);
}

void display(int arr[], int size) {
	int i;
	for (i = 0; i < size; i++) {
		printf("%d ", arr[i]);
	}
	printf("\n");
}

int main(void) {
	int source;
	struct Graph *g = (struct Graph *)malloc(sizeof(struct Graph));
	printf("Enter number of vertices : ");
	scanf("%d",&g->V);
	printf("Enter number of edges : ");
	scanf("%d",&g->E);
	g->edge = (struct Edge *)malloc(g->E * sizeof(struct Edge));
	for(int i=0;i<g->E;i++)
	{
		printf("Enter edge %d properties Source, destination, weight respectively\n",i+1);
		scanf("%d",&g->edge[i].u);
		scanf("%d",&g->edge[i].v);
		scanf("%d",&g->edge[i].w);
	}
	printf("Enter the source node : ");
	scanf("%d",&source);
	bellmanford(g,source );
	return 0;
}
```

Output:


"C:\Users\dmell\OneDrive\Desktop\Subjects\AOA\Bellman Ford Algorithm.exe"
Enter number of vertices : 5
Enter number of edges : 10
Enter edge 1 properties Source, destination, weight respectively
0 1 6
Enter edge 2 properties Source, destination, weight respectively
0 2 7
Enter edge 3 properties Source, destination, weight respectively
1 2 8
Enter edge 4 properties Source, destination, weight respectively
1 4 -4
Enter edge 5 properties Source, destination, weight respectively
1 3 5
Enter edge 6 properties Source, destination, weight respectively
3 1 -2
Enter edge 7 properties Source, destination, weight respectively
2 3 -3
Enter edge 8 properties Source, destination, weight respectively
2 4 9
Enter edge 9 properties Source, destination, weight respectively
4 0 2
Enter edge 10 properties Source, destination, weight respectively
4 3 7
Enter the source node : 0
Distance array: 0 2 7 4 -2
Predecessor array: 0 3 0 2 1
Process returned 0 (0x0) execution time : 65.722 s
Press any key to continue.

PAGE No
DATE / /

8863

Experiment 10 : Bellman Ford.

Post lab

Q1

① Dynamic Programming approach is similar to divide and conquer in breaking down the problem into smaller and smaller sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, these smaller sub-problems results are remembered and used for similar or overlapping sub problems

② Dynamic Programming is used when we have problems, which can be divided into similar sub-problem, dynamic algorithm will try to examine results of previously solved sub-problems. The solutions of sub-problem are combined in order to achieve best solution

③ ~~It is used for when~~ Therefore

a) ~~Problem~~ The problem should be able to be divided into smaller overlapping sub problem

b) An optimum solution can be achieved by using an optimum solution of smaller sub problems

c) It uses Memoization.