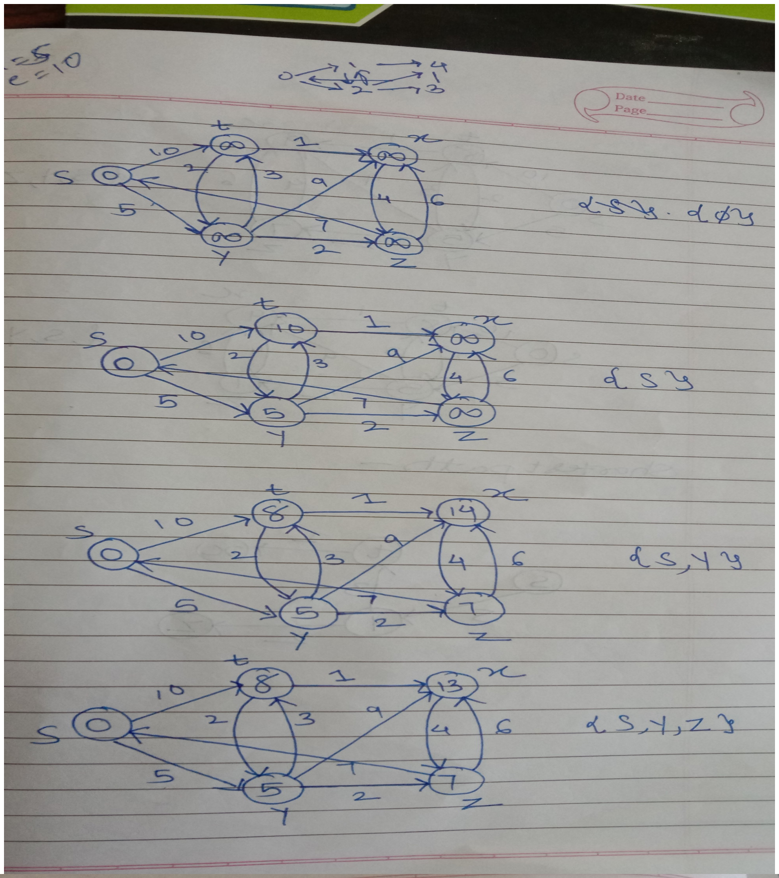
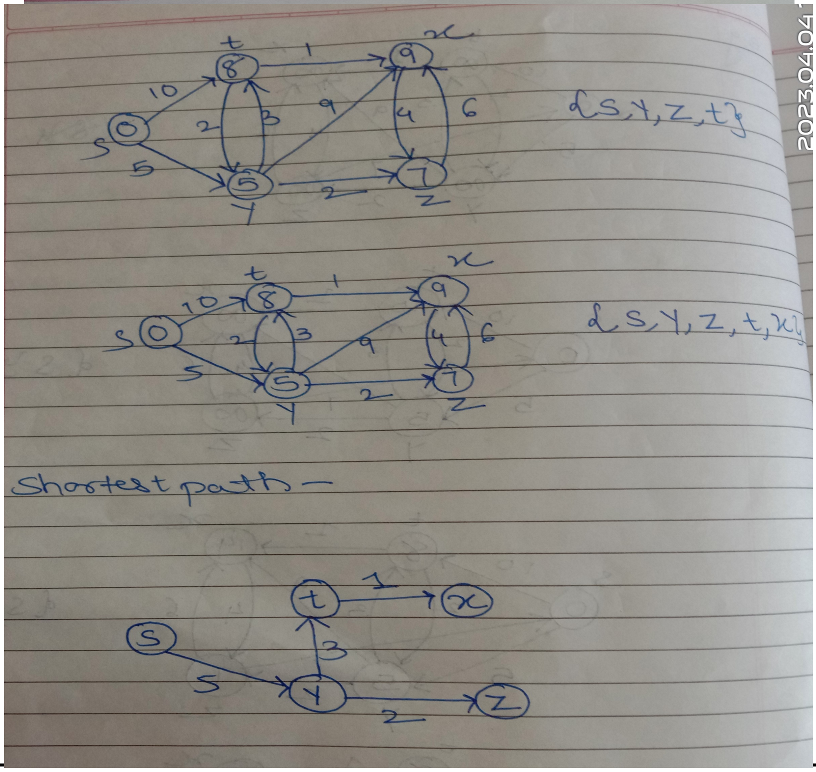
NAME:	Bhavesh Prashant Chaudhari
UID:	2021300018
SUBJECT	DAA
EXPERIMENT NO :	05
AIM:	To find shortest path from single source using Dijkstras Algorithm.
PROBLEM STATEMENT 1:	
THEORY	Dijkstra’s algorithm is a popular algorithms for solving many single-source shortest path problems having non-negative edge weight in the graphs i.e., it is to find the shortest distance between two vertices on a graph. The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source. It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited. The need for Dijkstra’s algorithm arises in many applications where finding the shortest path between two points is crucial. For example, It can be used in the routing protocols for computer networks and also used by map systems to find the shortest path between starting point and the Destination. Dijkstra algorithm cannot be applied on graphs with negative edges.It works on both directed and undirected graphs.

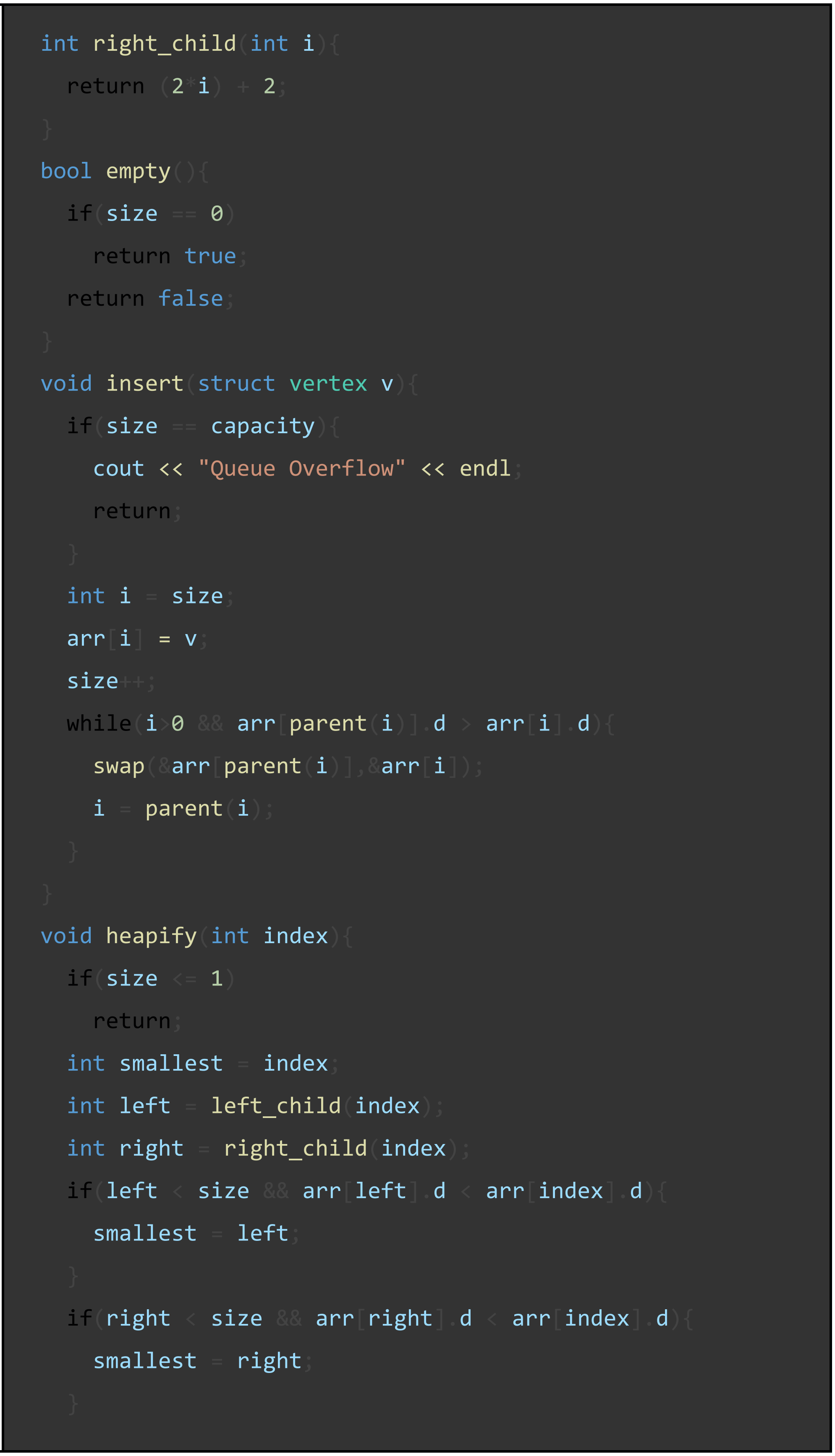
	Solved Example-  v=5
	e=10
	Date
	Page
	{S} . {∅}
	{S}
	{S, Y}
	{S, Y, Z}  Shortest path –

ALGORITHM

```
DIJKSTRA(G,W,S)
  S={}
  Q={}
  for each vertex u in G.V
    INSERT(Q,u)
  while(Q is not empty)
    u = EXTRACT-MIN(Q,u)
    S = S U {u}
    for each vertex v in G.Adj[u]
      RELAX(u,v,w)
      if the call of relax decreased v.d then
      DECREASE-KEY(Q,v,v.d)


RELAX(u,v,w)
if v.d > u.d + w(u,v)
  v.d = u.d + w(u,v)
  v.pred = u
```

PROGRAM:

```
#include<iostream>
#include<climits>


using namespace std;


const int  N = 20;
int edges[N][N];


//Structure that contains all necessary attr of a vertex
struct vertex{
  int name = -1;
  bool visited = false;
  int d = INT_MAX;
  int pred = -1;
};


//Helper function for swapping
void swap(struct vertex *x,struct vertex *y){
  struct vertex temp = *x;
  *x = *y;
  *y = temp;
}


//Implementing Priority using Min Heap
class PriorityQueue{
  private:
  struct vertex *arr;
  int size;
  int capacity;


  public:
  PriorityQueue(int n){
    arr = new struct vertex[n];
    size = 0;
    capacity = n;
  }
  int parent(int i){
    return (i-1)/2;
  }
  int left_child(int i){
    return (2*i) + 1;
  }
```

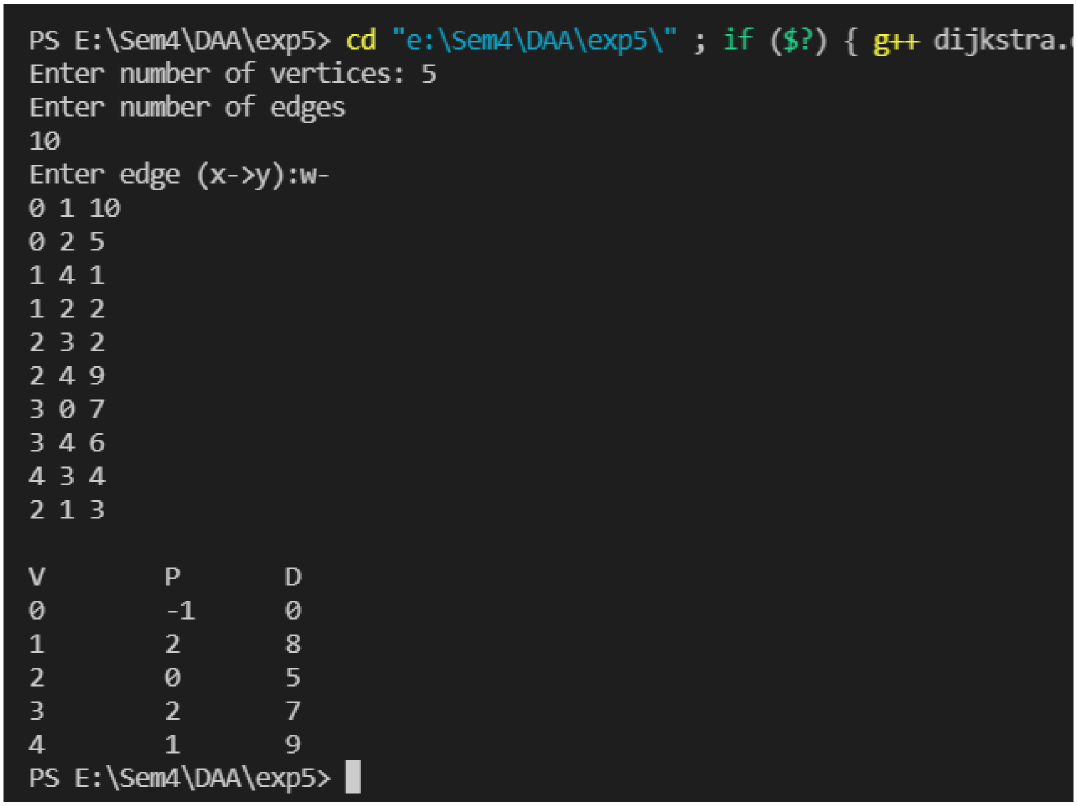


```
int right_child(int i){
  return (2*i) + 2;
}
bool empty(){
  if(size == 0)
    return true;
  return false;
}
void insert(struct vertex v){
  if(size == capacity){
    cout << "Queue Overflow" << endl;
    return;
  }
  int i = size;
  arr[i] = v;
  size++;
  while(i>0 && arr[parent(i)].d > arr[i].d){
    swap(&arr[parent(i)],&arr[i]);
    i = parent(i);
  }
}
void heapify(int index){
  if(size <= 1)
    return;
  int smallest = index;
  int left = left_child(index);
  int right = right_child(index);
  if(left < size && arr[left].d < arr[index].d){
    smallest = left;
  }
  if(right < size && arr[right].d < arr[index].d){
    smallest = right;
  }
```

```
    if(smallest != index){
      swap(&arr[smallest],&arr[index]);
      heapify(smallest);
    }
    return;
  }
  int extract_min(){
    if(size == 0){
      cout << "Queue Empty" << endl;
      return -1;
    }
    int root = arr[0].name;
    swap(&arr[size-1],&arr[0]);
    size--;
    heapify(0);
    return root;
  }
  void decrease_key(int name,int d){
    int i = 0;
    while(name != arr[i].name)
      i++;
    arr[i].d = d;
    while(i>0 && arr[parent(i)].d > arr[i].d){
      swap(&arr[parent(i)],&arr[i]);
      i = parent(i);
    }
  }
};


//Helper function to relax a vertex
int relax(int u,int v,struct vertex *vertices){
  if(vertices[v].d > vertices[u].d + edges[u][v]){
    vertices[v].d = vertices[u].d + edges[u][v];
    vertices[v].pred = u;
    return vertices[v].d;
  }
  return 0;
}



//Dijkstra function to find shortes path
void dijkstra(int src,int n,struct vertex *vertices){
  PriorityQueue Q(n);
  vertices[src].d = 0;
  vertices[src].name = src;
  for(int i=0;i<n;i++){
    vertices[i].name = i;
    Q.insert(vertices[i]);
  }
  while(!Q.empty()){
    int u = Q.extract_min();
    vertices[u].visited = true;
    for(int i=0;i<n;i++){
      if(edges[u][i] == 0)
        continue;
      int d = relax(u,i,vertices);
      if(d > 0){
        Q.decrease_key(i,d);
      }
    }
  }
}



//To display attr of all vertices
```

```
void display(struct vertex *v,int n){
  cout << "V \t P \t D" << endl;
  for(int i=0;i<n;i++){
    cout << i << " \t " << v[i].pred << " \t " << v[i].d << endl;
  }
}


int main(){
  int n;
  cout << "Enter number of vertices: ";
  cin >> n;
  struct vertex vertices[n];
  int edges_count;
  cout << "Enter number of edges" << endl;
  cin >> edges_count;
  cout << "Enter edge (x->y):w- " << endl;
  for(int i=0;i<edges_count;i++){
    int x,y,w;
    cin >> x;
    cin >> y;
    cin >> w;
    edges[x][y] = w;
  }
  dijkstra(0,n,vertices);
  cout << endl;
  display(vertices,n);
}
```

RESULT (SNAPSHOT)



```
PS E:\Sem4\DAA\exp5> cd "e:\Sem4\DAA\exp5\" ; if ($?) { g++ dijkstra.
Enter number of vertices: 5
Enter number of edges
10
Enter edge (x->y):w-
0 1 10
0 2 5
1 4 1
1 2 2
2 3 2
2 4 9
3 0 7
3 4 6
4 3 4
2 1 3

V       P       D
0       -1      0
1       2       8
2       0       5
3       2       7
4       1       9
PS E:\Sem4\DAA\exp5> █
```

CONCLUSION:	Through this experiment I understood what dijkstras algorithm is and how to implement it.Also learned about its application in real life.It cannot be used on graph with negative weight cycles reachable from source.