Answer Script

Question No. 01

Problem-1

a) Write down the advantages of bfs and dfs (At least 2)?
b) Write down the disadvantages of bfs and dfs (At least 1)? **10**

Answer No. 01

a) The advantages of BFS:

i. Always finds optimal solutions.
ii. Used to find the shortest path between vertices.
iii. Find the closest goal in less time.
iv. There is nothing like a useless path in BFS,since it searches level by level.

The advantages of DFS:

i. The memory requirement is Linear WRT Nodes.
ii. If the solution is far away then it consumes time.
iii. The solution can be found out without much more search.
iv. Finds the larger distant element in less time.

b) The disadvantages of BFS:

i. If a solution is far away then it consumes time.
ii. It can be very memory intensive since it needs to keep track of all the nodes in the search tree.
iii. It can be slow since it expands all the nodes at each level before moving on to the next level.

The disadvantages of DFS:

i. Determination of depth until the search has proceeded.
ii. Not Guaranteed that it will give you a solution.
iii. Cut-off depth is smaller so time complexity is more.

Question No. 02

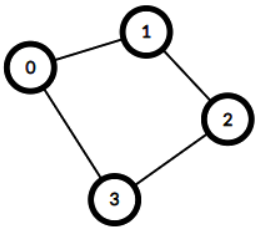
Problem-2

What are the different ways to represent a graph? Describe With Example. **10**

Answer No. 02

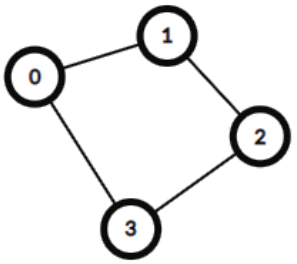
Graphs can be represented in different ways. Here are some commonly used ways to represent a graph:

1. **Adjacency Matrix:**
   An adjacency matrix can be thought of as a table with rows and columns. The row labels and column labels represent the nodes of a graph. An adjacency matrix is a square matrix where the number of rows, columns and nodes are the same. Each cell of the matrix represents an edge or the relationship between two given nodes. For example, adjacency matrix Aij represents the number of links from i to j, given two nodes i and j.



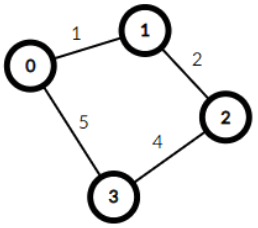

```
   0 1 2 3
---------------
0 | 0 1 0 1
1 | 1 0 1 0
2 | 0 1 0 1
3 | 1 0 1 0
```

2. **Adjacency List:**
   In the adjacency list representation of a graph, every vertex is represented as a node object. The node may either contain data or a reference to a linked list. This linked list provides a list of all nodes that are adjacent to the current node. Consider a graph containing an edge connecting node 0 and node 1. Then, node 0 will be available in node 1's linked list.



0 -> {1, 3}
1 -> {0, 2}
2 -> {1, 3}
3 -> {0, 2}

3. **Edge List:**
   An edge list is a list of all edges in the graph. Each edge is represented as a tuple (u, v, w), where u and v are the vertices at the endpoints of the edge and w is the weight of the edge.



[
(0, 1, 1),
(1, 2, 2),
(2, 3, 4),
(3, 0, 5)
]

Question No. 03

Problem-3

Write down a c++ program to detect cycle in an **undirected** graph.

15

Sample Input-	**Sample Output-**
4 3 1 2 2 3 1 3 Explanation: Here,Number of node is 4 Number of edge is 3	Cycle Exist
3 2 1 2 2 3 Explanation: Here,Number of node is 3 Number of edge is 2	No Cycle

Answer No. 03

Code:

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5;
int visited[N];
vector<int> adj_list[N];

bool detect_cycle(int node, int parent)
{
   visited[node] = 1;

   for(auto adj_node: adj_list[node])
   {
      if(visited[adj_node] == 0)
      {
         bool got_cycle = detect_cycle(adj_node, node);
```

```
            if(got_cycle)
                return true;

        }
        else if(adj_node != parent)
        {
            return true;
        }
    }

    visited[node] = 2;
    return false;
}

int main()
{
    int n, m;
    cin>>n>>m;
    for(int i=0; i<m; i++)
    {
        int u, v;
        cin>>u>>v;

        adj_list[u].push_back(v);
        adj_list[v].push_back(u);
    }
    bool cycle_exists = false;
    for(int i=1; i<=n; i++)
    {
        if(visited[i] == 0)
        {
            bool got_cycle = detect_cycle(i, -1);
            if(got_cycle)
            {
                cycle_exists = true;
                break;
            }
        }
    }
    if(cycle_exists)
        cout<<"Cycle Exist\n";
    else
        cout<<"No Cycle\n";
    return 0;
}
```

Question No. 04

Problem-4

You are given a positive integer n. The next line will contain n positive integers .Now calculate the total sum of the array.
Implement it using recursion.

Constraints-

1<=n<=100 , 1<=A[i]<=1000

Write a C++ program for this problem **10**

Sample Input -	**Sample Output -**
4 26 3 17 5	51

Answer No. 04

Code:

```
#include<bits/stdc++.h>
using namespace std;
int totalSum(vector<int> v, int n)
{
   if(n==1)
      return v[0];
   else
      return v[n-1] + totalSum(v, n-1);
}

int main()
{
   int n;
   cin>>n;
   vector<int> v(n);
   for(int i=0; i<n; i++)
      cin>>v[i];

   int sum = totalSum(v, n);
   cout<<sum<<"\n";

   return 0;
}
```

Question No. 05

Problem-5

Bangladesh has n cities, and m roads between them.You can go from one city to another if there exists a path between those two cities.
The goal is to reach from city 1 to n.

Input -
The first input line has two integers n and m the number of cities and roads. The cities are numbered 1,2,…,n .After that, there are m lines describing the roads. Each line has two integers a and b. There is a road between those cities.A road always connects two different cities, and there is at most one road between any two cities.

Output - Print "YES" if your goal is possible, and "NO" otherwise.

Constraints-
2<=n<=10^5 , 1<=m<=2*10^5 , 1<=a,b<=n
Write a C++ program for this problem **15**

Sample Input-	**Sample Output-**
10 8 1 3 3 4 3 6 4 6 2 5 1 7 3 10 9 8	YES
8 6 7 4 7 6 4 6 2 5 1 3 7 8	NO

Answer No. 05

Code:

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5 + 5;
vector<int> adj_list[N];
int n,m;
int visited[N];

bool DFS(int node)
{
   if(node == n)
      return true;
   visited[node] = 1;
   for(auto adj_node: adj_list[node])
   {
      if(!visited[adj_node])
      {
         if(DFS(adj_node))
            return true;;
      }
   }
   return false;
}

int main()
{
   cin>>n>>m;
   for(int i=0; i<m; i++)
   {
      int u, v;
      cin>>u>>v;
      adj_list[u].push_back(v);
      adj_list[v].push_back(u);
   }
   int src = 1;
   bool goal = DFS(src);
   if(goal)
      cout<<"YES\n";
   else
      cout<<"NO\n";
   return 0;
}
```

Question No. 06

Problem-6

You and some monsters are in a labyrinth.Your goal is to reach from cell A to one of the safe boundary cell.You can walk left, right, up and down.But you can't go to those cells , if there is a monster in that cell or the cell contains a wall.You can only go to the safe(.) cell.

Input -

The first input line has two integers n and m the height and width of the map. After this there are n lines of m characters describing the map. Each character is .(safe cell), # (wall), A (start), or M (monster). There is exactly one A in the input.

Output -

First print "YES" if your goal is possible, and "NO" otherwise.
If your goal is possible, also print any valid path(the length of the path and its description using characters `D`, `U`, `L`, and `R`).

Constraints-

$1 \le n,m \le 1000$

Write a C++ program for this problem **15**

Sample Input -	**Sample Output -**
5 8 ..###### #M..A..# #.#.M#.# #M#..#.. ..#.####	YES 5 RRDDR

Explanation -

For the above map , the following are the boundary cells
(1,1),(1,2),(1,3),(1,4)(1,5),(1,6), (1,7), (1,8)
(1,1),(2,1),(3,1),(4,1),(5,1)
(1,8),(2,8),(3,8),(4,8),(5,8)
(5,1),(5,2),(5,3),(5,4),(5,5),(5,6),(5,7),(5,8)
and,
safe boundary cells are (1,1) ,(1,2),(5,1),(5,2),(5,4),(4,8)

Answer No. 06

Code:

```
#include<bits/stdc++.h>
using namespace std;
const int maxN = 1000;
char grid[maxN][maxN];
bool visited[maxN][maxN];
pair<int, int> parent[maxN][maxN];
int dx[4] = {1, 0, -1, 0};
int dy[4] = {0, 1, 0, -1};
char dir[4] = {'D', 'R', 'U', 'L'};

void BFS(int x, int y, int n, int m)
{
   queue<pair<int, int>> q;
   q.push({x, y});

   visited[x][y] = true;
   parent[x][y] = {-1, -1};

   while(!q.empty())
   {
      pair<int, int> u = q.front();
      q.pop();

      for(int i=0; i<4; i++)
      {
         int dx_, dy_;
         dx_ = u.first + dx[i];
         dy_ = u.second + dy[i];

         if(dx_>=1 && dx_<=n && dy_>=1 && dy_<=m && !visited[dx_][dy_] && grid[dx_][dy_]
!= '#' && grid[dx_][dy_] != 'M')
         {
            visited[dx_][dy_] = true;
            q.push({dx_, dy_});
            parent[dx_][dy_] = u;
         }

      }
   }
}

void printSolution(int end_x, int end_y)
```

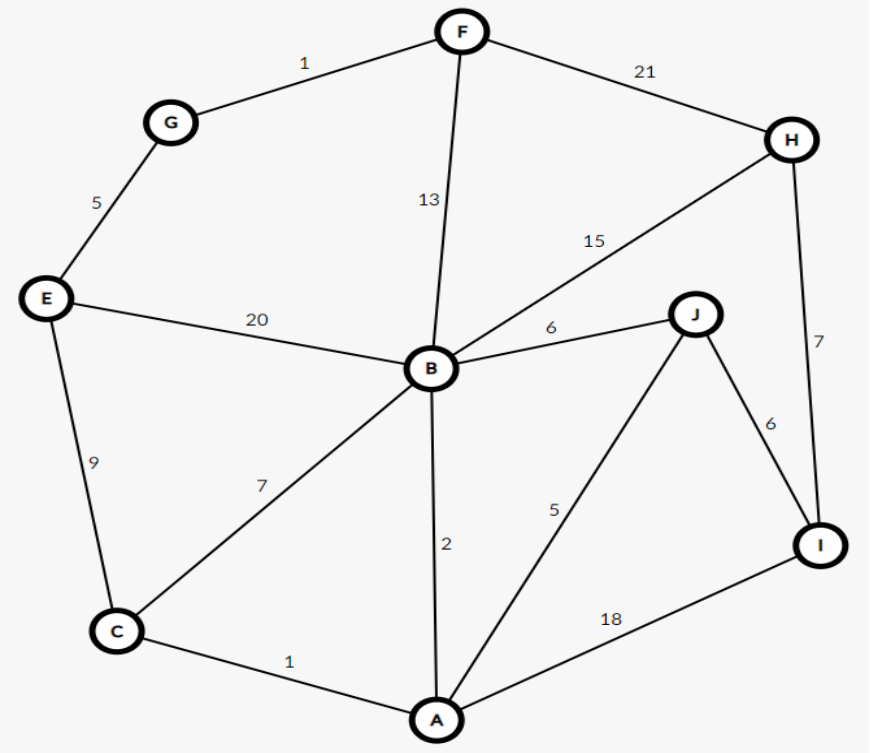
```
{
    cout<<"YES\n";

    vector<pair<int, int>> path;
    path.push_back({end_x, end_y});

    int x = end_x, y = end_y;

    while(parent[x][y] != make_pair(-1,-1))
    {
        pair<int, int> p = parent[x][y];
        x = p.first, y = p.second;
        path.push_back({x, y});
    }

    reverse(path.begin(), path.end());

    string ans;

    for(int i=0; i<path.size()-1; i++)
    {
        for(int j=0; j<4; j++)
        {
            int dx_, dy_;
            dx_ = path[i].first + dx[j];
            dy_ = path[i].second + dy[j];

            if(dx_==path[i+1].first && dy_==path[i+1].second)
            {
                ans.push_back(dir[j]);
                break;
            }
        }
    }
    cout<<ans.size()<<"\n";
    cout<<ans<<"\n";
}


int main()
{
    int n, m;
    cin>>n>>m;
    int start_x, start_y, end_x, end_y;
```

```
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=m; j++)
        {
            cin>>grid[i][j];
            if(grid[i][j] == 'A')
            {
                start_x = i;
                start_y = j;
            }
        }
    }
    BFS(start_x, start_y, n, m);

    for(int i=1; i<=n; i++)
    {
        if(grid[i][1]!='#' && grid[i][1]!= 'M' && visited[i][1])
        {
            printSolution(i, 1);
            return 0;
        }
        else if(grid[i][m]!='#'  && grid[i][m]!='M' && visited[i][m])
        {
            printSolution(i, m);
            return 0;
        }
    }

    for(int i=1; i<=m; i++)
    {
        if(grid[1][i]!='#' && grid[1][i]!= 'M' && visited[1][i])
        {
            printSolution(1, i);
            return 0;
        }
        else if(grid[n][i]!='#'  && grid[n][i]!='M' && visited[n][i])
        {
            printSolution(n, i);
            return 0;
        }
    }

    cout<<"NO\n";
    return 0;
}
```

Question No. 07

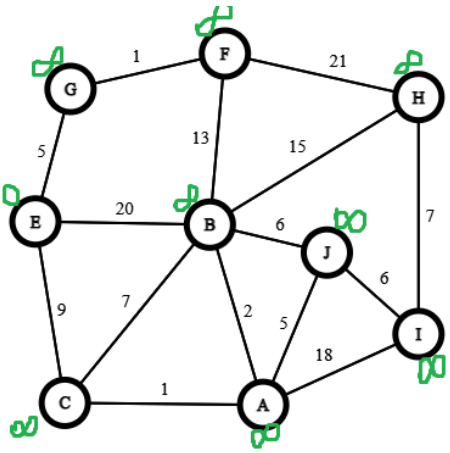
Problem-7

Write the shortest distance from node E to every other node using the Dijkstra algorithm (the optimized version) for the following graph .You need to write all the steps. **15**



Answer No. 07

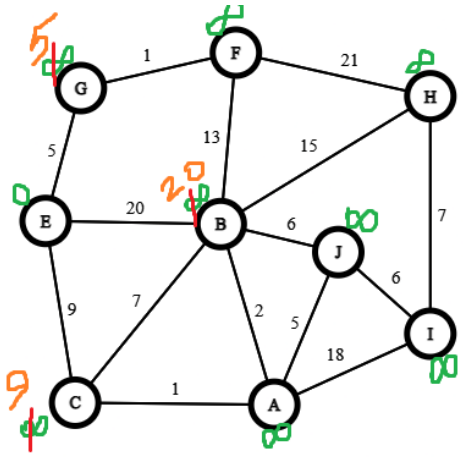
Step: 1



Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞

First declare every node cost infinity. Here, E is the starting node. That's why E node cost is equal to 0 and push E node and its cost in the priority queue. Here we use priority queue min heap. Where root has minimum value.

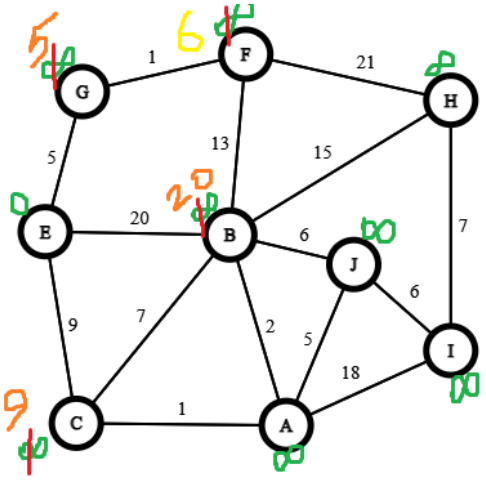
Step: 2



				X					
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞

Node E adjacent nodes are B, C and G. Initially each node's cost is infinity. E node cost 0. E to G total cost is 0+5=5. Which is less than G previous infinity cost. Then relax the G node costs infinity by 5. Now G new cost is 5. E to C total cost is 0+9=9. Which is less than C previous infinity cost. Then relax C nodes cost infinity by 9. Now C new cost is 9. Same way, E to B total cost is 0+20=20. Which is less than B previous infinity cost. Then relax B nodes cost infinity by 20. Now B new cost is 20.
Now, Node B total cost is 20, C Node total cost is 9 and G node total cost is 5 which is the lowest cost compared to the B and C node.
Pop the priority queue previous top element which is E node cost and push G node cost value in the priority queue. Then push C node with its cost and then push B node with its cost.

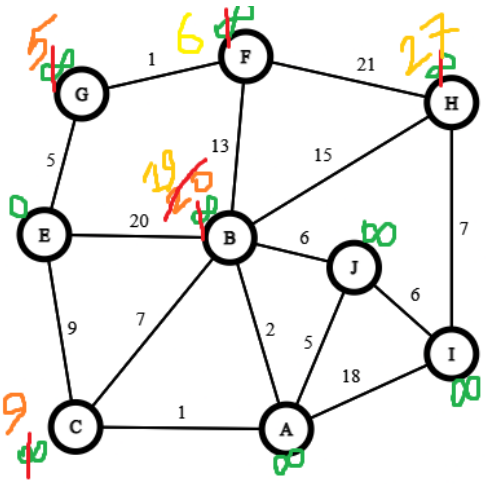
Step: 3



				X		X			
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞

Now the priority queue top node is G node. Its adjacent node is F node. Now G to F node total cost is 5+1=6. Which is lower than the previous infinity cost. Now relax this node costs infinity by 6 and push this node in the priority queue and pop priority queue top node.
Now, the Priority queue has nodes F, C and B.

Step: 4



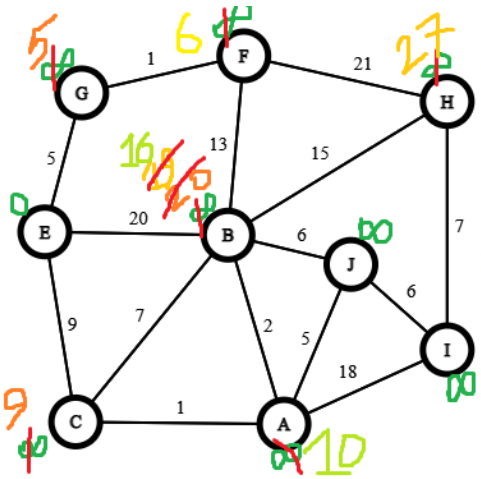
				X	X	X			
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞

Now, the priority queue top node is F and its adjacent nodes are H and B. Now F to H total cost is 6+21=27. Which is lower than the previous infinity cost. Now relax this node costs infinity by 27. Now H node cost is 27. F to B total cost is 6+13=19. Which is lower than the previous cost. Now relax this node's previous cost by 19. Now B node's new cost is 19.

Pop the priority queue previous top element which is F node cost and push B node cost value in the priority queue and then push H node with its cost.

Now, the Priority queue has nodes C,B and H.

Step: 5



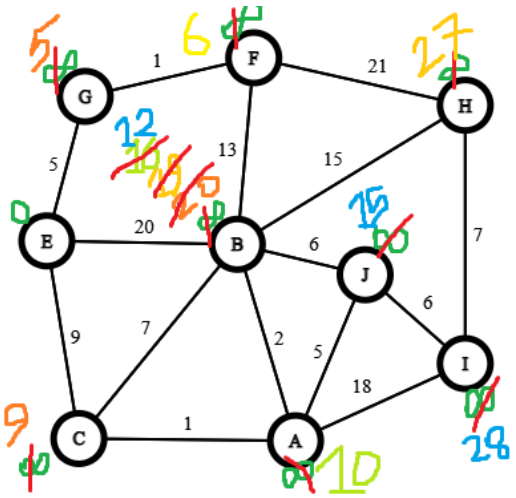
			X	X	X	X			
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞
C	10	16	9	0	6	5	27	∞	∞

Now, the priority queue top node is C and its adjacent nodes are A and B. Now C to A total cost is 9+1=10. Which is lower than the previous infinity cost. Now relax this node costs infinity by 10. Now A node cost is 10. C to B total cost is 9+7=16. Which is lower than the previous cost. Now relax this node's previous cost by 16. Now B node's new cost is 16.

Pop the priority queue previous top element which is C node cost and push A node cost value in the priority queue and then push B node with its new updated cost.

Now, the Priority queue has nodes A, B and H.

Step: 6

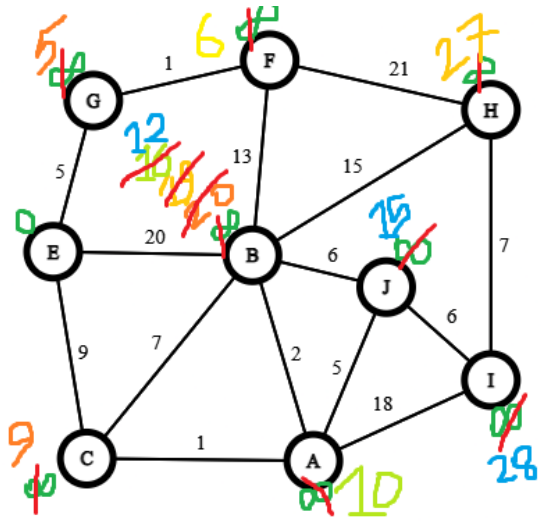


	X		X	X	X	X			
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞
C	10	16	9	0	6	5	27	∞	∞
A	10	12	9	0	6	5	27	28	15

Now, the priority queue top node is A and its adjacent nodes are I, J and B. Now A to I total cost is 10+18=28. Which is lower than the previous infinity cost. Now relax this node costs infinity by 28. Now I node cost is 28. A to B total cost is 10+2=12. Which is lower than the previous cost. Now relax this node's previous cost by 12. Now B node's new cost is 12. A to J total cost is 10+5=15. Which is lower than the previous infinity cost. Now relax this node's previous cost by 15. Now J node's new cost is 15.

Pop the priority queue previous top element which is A node cost and push B node cost value in the priority queue and then push J node with its new updated cost and last push J node with its new updated cost.
Now, the Priority queue has nodes B, J, H and I.

Step: 7

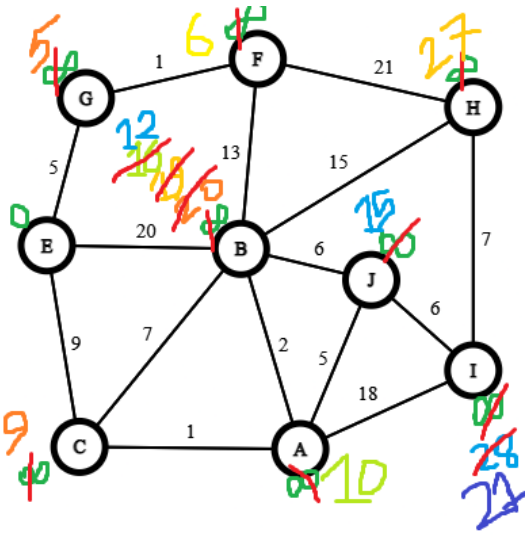


	X	X	X	X	X	X			
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞
C	10	16	9	0	6	5	27	∞	∞
A	10	12	9	0	6	5	27	28	15
B	10	12	9	0	6	5	27	28	15

Now, the priority queue top node is B and its adjacent nodes are H and J. Now B to J total cost is 12+6=18. Which is greater than the previous infinity cost. So, It don't need to relax. B to H total cost is 12+15=27. Which is equal to the previous cost. So, It also don't need it to relax.

Pop the priority queue previous top element which is B node cost.
Now, the Priority queue has nodes J, H and I.

Step: 8

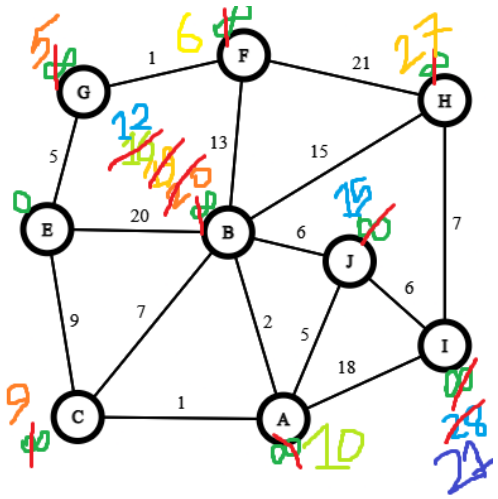


	X	X	X	X	X	X			X
Selected Node	A	B	C	E	F	G	H	I	J
---	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞
C	10	16	9	0	6	5	27	∞	∞
A	10	12	9	0	6	5	27	28	15
B	10	12	9	0	6	5	27	28	15
J	10	12	9	0	6	5	27	21	15

Now, the priority queue top node is J and its adjacent node is I. Now J to I total cost is 15+6=21. Which is lower than the previous infinity cost. Now relax this node's previous cost by 21. Now I node's new cost is 21.

Pop the priority queue previous top element which is J node cost.
Now, the Priority queue has nodes I and H.

Step: 9

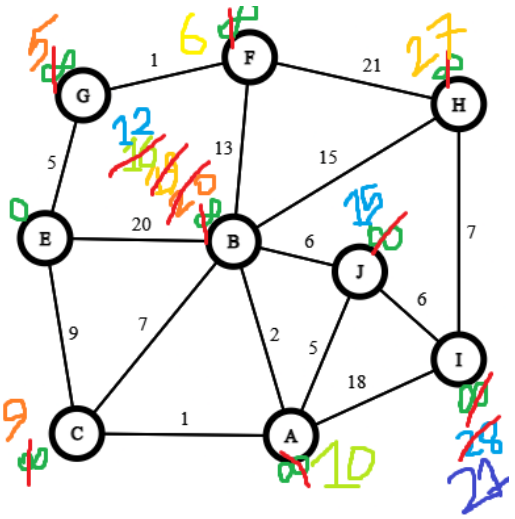


	X	X	X	X	X	X		X	X
Selected Node	A	B	C	E	F	G	H	I	J
---	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞
C	10	16	9	0	6	5	27	∞	∞
A	10	12	9	0	6	5	27	28	15
B	10	12	9	0	6	5	27	28	15
J	10	12	9	0	6	5	27	21	15
I	10	12	9	0	6	5	27	21	15

Now, the priority queue top node is I and its adjacent node is H. Now I to H total cost is 21+7=28. Which is greater than the previous node cost. So, It don't need to relax.

Pop the priority queue previous top element which is I node cost.
Now, the Priority queue has nodes H.

Step: 10



	X	X	X	X	X	X	X	X	X
Selected Node	A	B	C	E	F	G	H	I	J
---------	∞	∞	∞	0	∞	∞	∞	∞	∞
E	∞	20	9	0	∞	5	∞	∞	∞
G	∞	20	9	0	6	5	∞	∞	∞
F	∞	19	9	0	6	5	27	∞	∞
C	10	16	9	0	6	5	27	∞	∞
A	10	12	9	0	6	5	27	28	15
B	10	12	9	0	6	5	27	28	15
J	10	12	9	0	6	5	27	21	15
I	10	12	9	0	6	5	27	21	15
H	10	12	9	0	6	5	27	21	15

Now, the priority queue top node is H and its adjacent node is F, B and I which is already visited. Don't need to relax any node.
Pop the priority queue previous top element which is H node cost.
Now, the Priority queue is empty. It's means all node is visited.

Question No. 08

Problem-8

What is the recursive case in a recursive function and how does it relate to the base case? Explain it. **10**

Answer No. 08

In general, recursive functions consist of two parts: the base case and the recursive case.

In a recursive function, the recursive case is the part of the function that calls itself. It is the part of the function that breaks down the problem into smaller subproblems and solves them recursively until a base case is reached.

For example, consider the following recursive function to calculate the factorial of a number:

```
int factorial(int n) {
  if (n == 0) {       // base case
    return 1;
  } else {            // recursive case
    return n * factorial(n - 1);
  }
}
```

In this function, the base case is when n is equal to 0, and the function returns 1. The recursive case is when n is greater than 0, and the function calls itself with n - 1 as the argument. The function breaks down the problem of calculating the factorial of n into the smaller subproblem of calculating the factorial of n - 1, and solves it recursively until the base case is reached.

The base case is the condition that stops the recursion and is solved without further recursion. It is the simplest case for which the solution is known or trivial. The base case is necessary to prevent the function from infinitely calling itself, which would result in a stack overflow error.

It is important to ensure that the recursive case eventually reaches the base case, otherwise the function will continue to call itself indefinitely, resulting in an infinite loop and a stack overflow error.