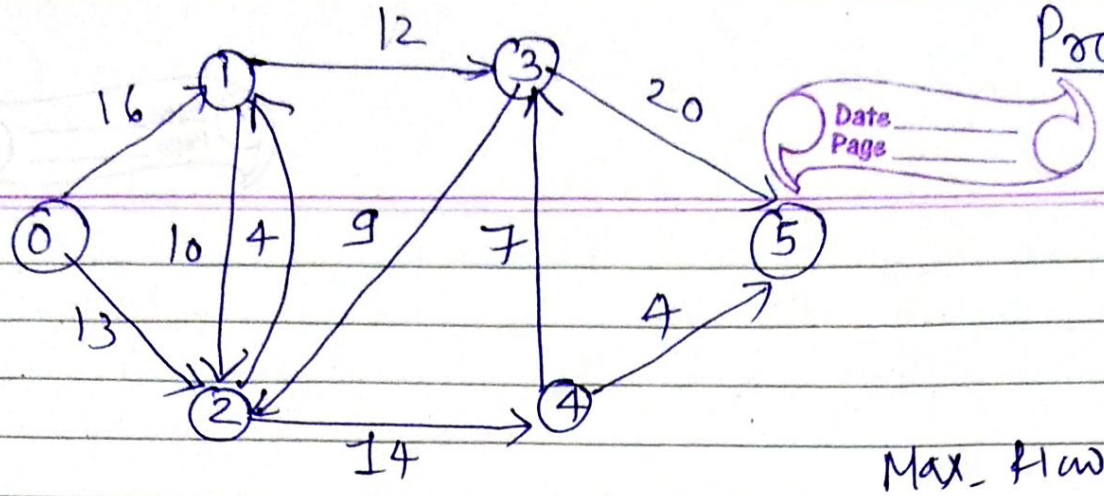
Practical-3

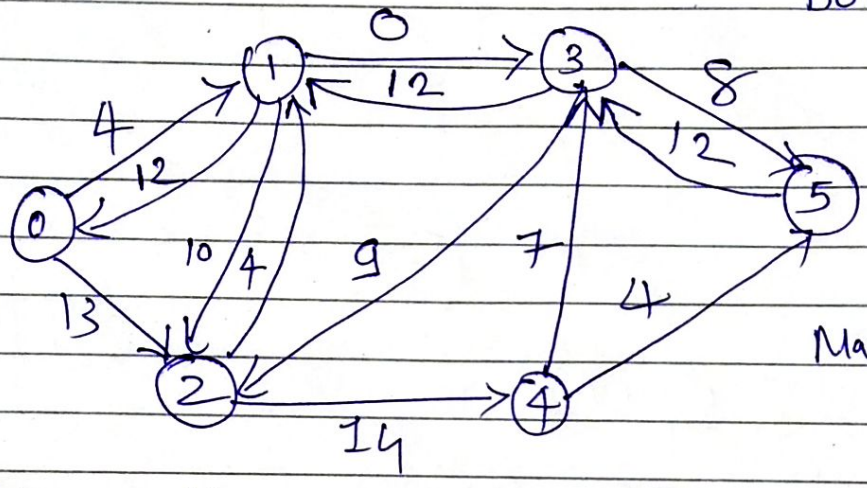
Ex



Max_flow = 0

(I) Augmenting path: 0-1-3-5

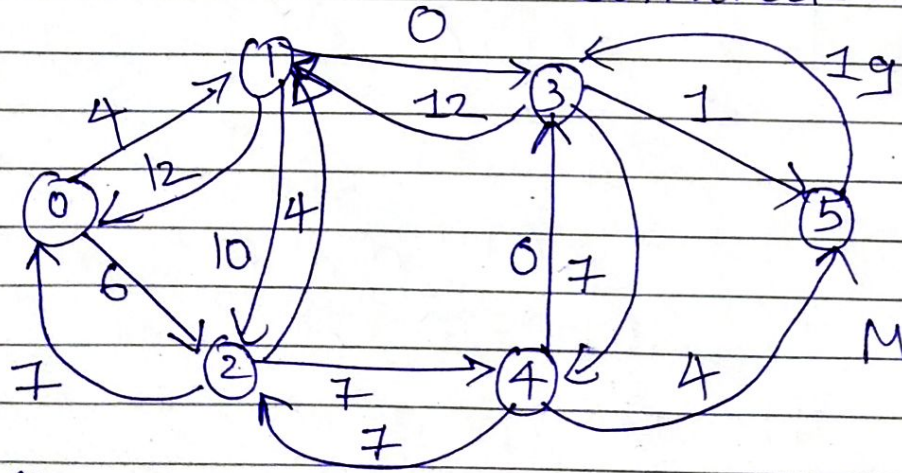
Bottleneck = 12



Max_flow = 0+12
= 12

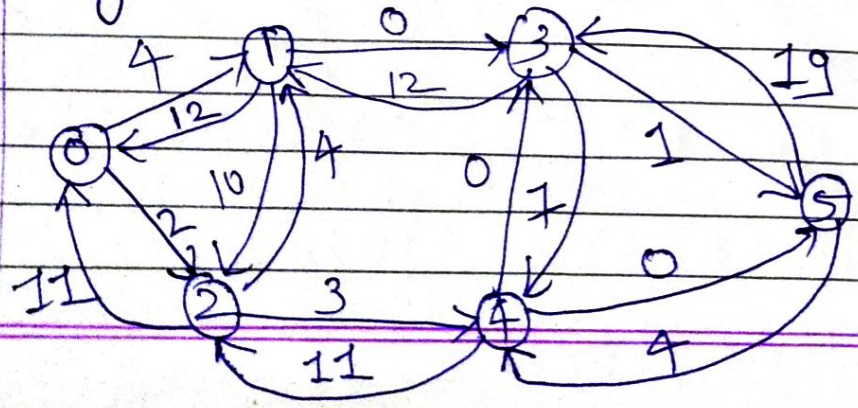
(II) Augmenting path: 0-2-4-3-5

Bottleneck = 7



Max_flow = 12+7
= 19

(III) Augmenting path: 0-2-4-5, Bottleneck : 4



Max_flow = 19+4
= 23

Now, we can see that there doesn't exist any augmenting path

$\therefore$ Max_flow = 23

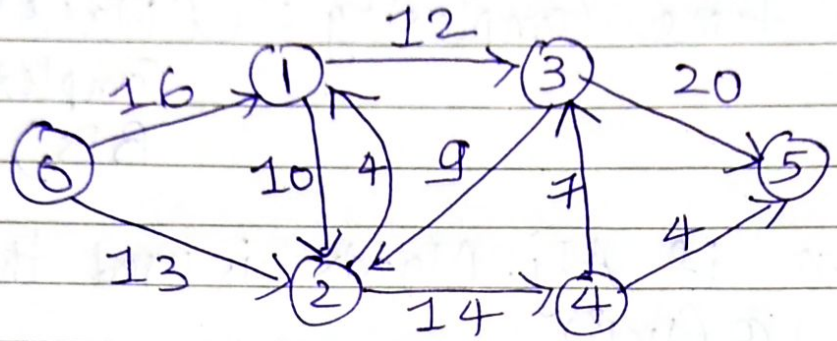
★ Worst case time Complexity : O(Max_flow * Complexity of BFS)

- We know if Adj. Matrix is used then BFS takes $O(|V|^2)$
- If Adj. List is used then BFS takes $O(|V|+|E|)$

Pseudo-code

Say $n = 6$

$G[6][6] = \{ \{0,16,13,0,0,0\}, \{0,0,10,12,0,0\}, \{0,4,0,0,14,0\}, \{0,0,9,0,0,20\}, \{0,0,0,7,0,4\}, \{0,0,0,0,0,0\} \}$



Call:

fordfulkerson (G, 0, 5)

↑ Source ↑ Sink

```
int fordfulkerson (int G[n][n], int s, int t)
{
→ Create rgraph[n][n]
  for (u=0; u<n; u++)
  { for (v=0; v<n; v++)
    { rgraph[u][v] = G[u][v];
    } }
→ Create parent[n] (Parent will be assigned by bfs())
  max_flow = 0;
→ while (bfs(rgraph, s, t, parent))
  { path_flow = ∞;
    for (v=t; v!=s; v= parent[v])
    { u = parent[v];
      path_flow = Min{path_flow, rgraph[u][v]};
    }
```

```
for (v=t; v!=s; v=parent[v])
{
    u=parent[v];
    rgraph[u][v] -= path_flow;
    rgraph[v][u] += path_flow;
}
```

end of while

```
    max_flow += path_flow;
    return max_flow;
}
```

☆

```
bool bfs(rgraph[n][n], s, t, parent[n])
{
    visited[i]=0, for every i= 0 to (n-1)
    Queue q; // Create Queue
    Enqueue(q,s);
    visited[s]=1;
    parent[s]=-1;
    while(q is not empty)
    {
        u=dequeue(q);
        for(v=0; v<n; v++)
        {
            if(visited[v]==0 && rgraph[u][v]>0)
            {
                if(v==t)
                {
                    parent[v]=u;
                    return true;
                }
                Enqueue(q,v);
                visited[v]=1;
                parent[v]=u;
            }
        }
    }
    return false;
```