4/18/19

Assignment-5

1. Given $n \times n$ matrix W.

```
FLOYD-WARSHALL (W, n)
  //Input: nxn matrix W
  //Output: matrix L of shortest path weights
  L = W
  for k=1 to n
    for i=1 to n
      for j=1 to n
        if L[i][j] > L[i][k] + L[k][j]
          L[i][j] = L[i][k] + L[k][j]
  return L.
```

The matrix L[..][..] from the above algorithm can be computed in $O(n^3)$ time. The worst case time complexity is computed from the three loops running through whole array of 1 to n. Thus the time complexity (running time) is $O(n^3)$.

5. Given language L_1 and L_2 are in P.
Show that $L_1 \cap L_2$ and $L_1 L_2$ are also in P.

* ~~Intersection~~ Intersection: Let $L_1, L_2 \in P$.

As $L_1 \in P$, $\exists$ Machine M_1 with time complexity of $O(n^{k_1})$ for some constant k_1.

Similarly, As $L_2 \in P$, $\exists$ Machine M_2 with time complexity of $O(n^{k_2})$ for some constant k_2

Let us consider a machine M, with polynomial time complexity for $L_1 \cap L_2$

M = 1. Given input 'x'

2. Run M

2.1. If M_1 accepted, then run M_2 on x,
~~2.2~~. else reject.

2.2. If M_2 also accepted, then accept,
else reject.

In worst case, M will run both M_1 and M_2, in which case it uses $O(n^{k_1}) + O(n^{k_2})$ steps. Let $k = \max(k_1, k_2)$. ~~When~~ Thus M has time complexity of $O(n^k)$ and therefore, $L(M) = L_1 \cap L_2$ where $L_1 \cap L_2 \in P$.

* Concatenation: Given L_1 and $L_2 \in P$.

Show that $L_1 L_2 \in P$.

We can deduce from this that M_1 and M_2 exists such that M_1 is decider for L_1 and has a time complexity $O(n^{k_1})$ and M_2 is a decider for L_2 and has time complexity of $O(n^{k_2})$ for constant k_1 and k_2

$\therefore L_1 L_2 = \{x_1 x_2 \mid x_1 \in L_1, x_2 \in L_2\}$

Let us divide the input 'x' into two parts x_1 and x_2. such that $x_1 \in L_1$ and $x_2 \in L_2$

1. Given input $x = a_1, a_2 \ldots a_n$
2. For $i = 0$ to n do
   - 2.1. Let $x_1 = a_1 \ldots a_i$ and $x_2 = a_{i+1} \ldots a_n$
   - 2.2. Run M_1 with input x_1
   - 2.3. Run M_2 with input x_2
   - 2.4. If both M_1 and M_2 are accepted then accept.
3. If no choice of x_1 and x_2 then reject.

We must now show that it has polynomial time complexity. The main loop is traversed $(n+1)$ times. If we run M_1 on substring x_1, this will take atmost $O(n^{k_1})$ steps. Similarly for x_2 it will take $O(n^{k_2})$ steps. In total it takes upto $O(n^{k_1}) + O(n^{k_2}) = O(n^k)$ where $k = \max(k_1, k_2)$. The whole decider thus uses $(n+1) . O(n^k) = O(n^{k+1})$ steps.

Hence $L_1 L_2 \in P$.

2) There are 'n' undergraduate students
'k' departments at university

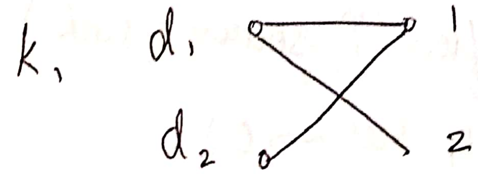
$\Rightarrow$ k_1 freshman, k_2 sophomores, k_3 juniors, k_4 seniors

where $k_1 + k_2 + k_3 + k_4 = k$.

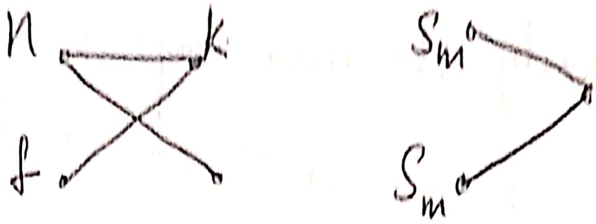
Let $k_1 = k_2 = k_3 = k_4 = 1$

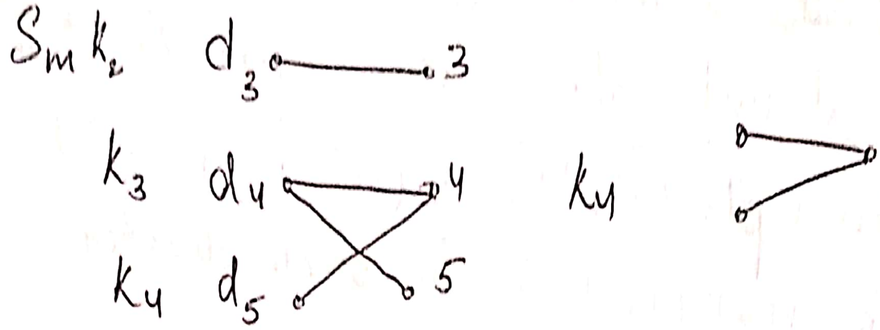
* A solution can be students (k), senate (k)

of



Students are picked from





The maximum matching is the solution. So the senate can be obtained by matching $d_1 2$, $d_2 1$, $d_3 3$, $d_4 5$, $d_3 4$

* Each selected student belongs to different department

Given Input : An 2D array $L[n][k]$ with n rows and k colomns (n - no. of students k - no. of department)

Output : 0 or 1 (True or False)

⇒ from array $P[k][k]$

res = maxflow(P, source, sink)

if(res → 0)

return 1

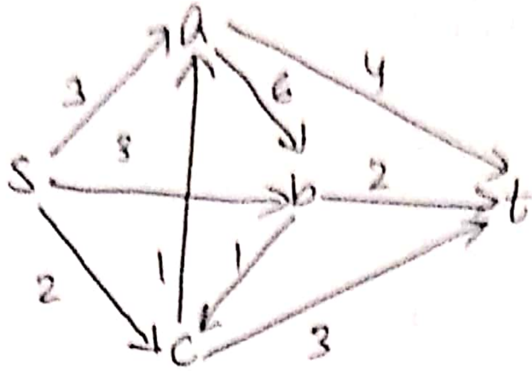
else

return 0

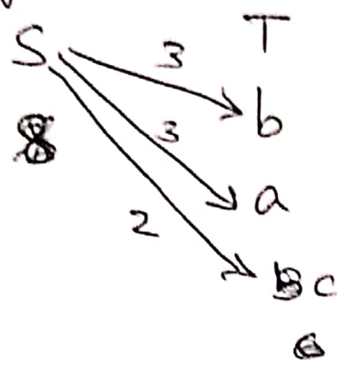
```
maxflow (P, s, t)
// i/p: P[k][k]
// o/p: max flow
for i = 0 to k
    for j = 0 to k
        A[i][j] = P[i][j]

maxflow = 0;
while (BFS (A, s, t, root)
    p_flow = 9999
    j = t.
    while (j != s)
        i = root [j]
        p_flow = min (p_flow, A [i][j])
        j = root [v]
    j = t
    while (j != s)
        j = root [j]
        A [i] [j] = p_flow
        A [i] [j] += p_flow
            j = root [j]
    max flow = maxflow + p_flow
return maxflow;
```

Scanned by CamScanner

3) Given.

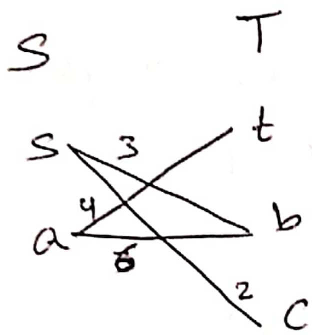


Performing cut 1:



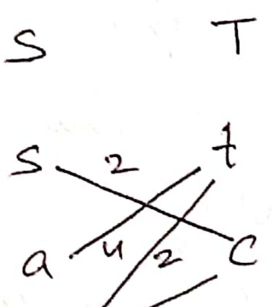
Capacity: 8

performing cut 2:



Capacity: 15

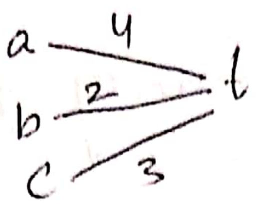
performing cut 3:



Capacity: 9

pefoming cut 4:

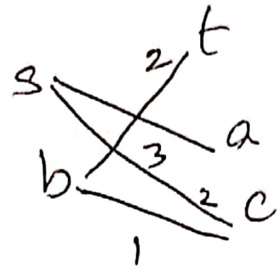
S T



Capacity: 9

peforming cut 5:

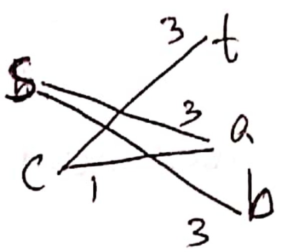
S T



Capacity: 8

peforming cut 6:

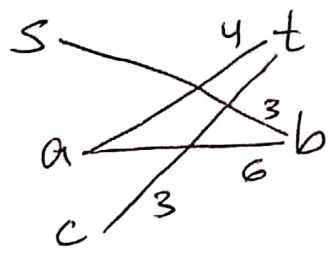
S T



Capacity: 10

pefoming cut 7:

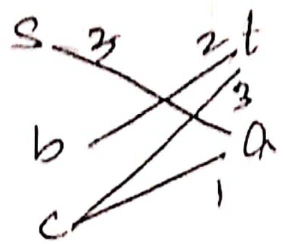
S T



Capacity: 16

performing cut 8 :

S T

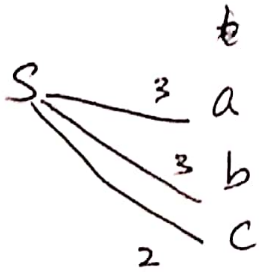


Capacity : 9

3b) Capacity = 8 has the minimum cuts

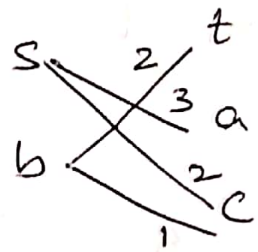
Cut 1 :

S T



Cut 5 :

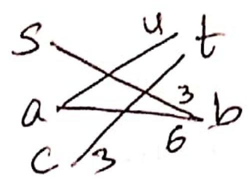
S T



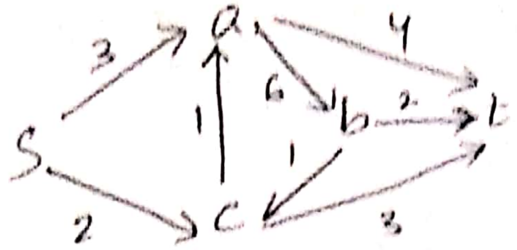
Capacity = 16 has the maximum cut.

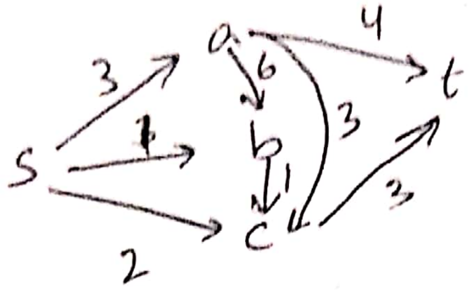
Cut 7 :

S T



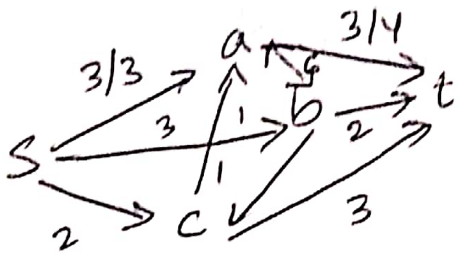
3c)



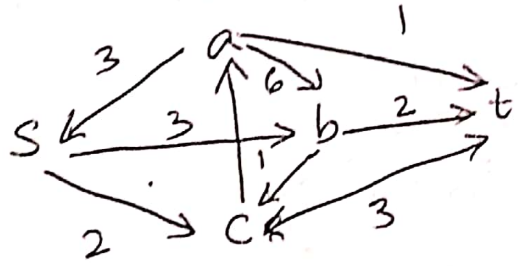


path: $s \to a \to t$

(i)

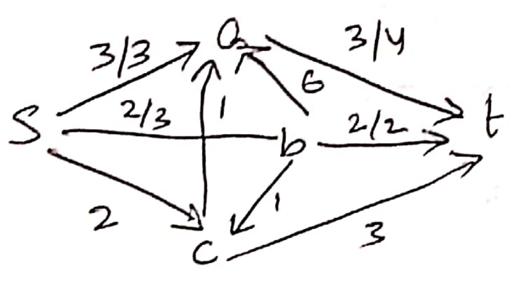


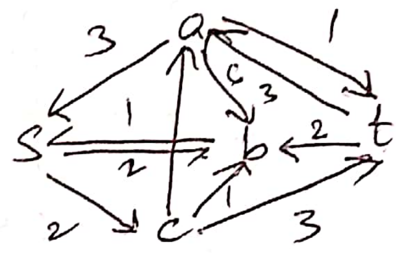
$\phi = 3$



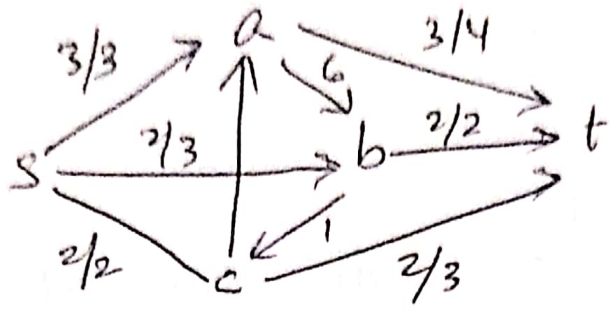
(ii) path: $s \to b \to t$.

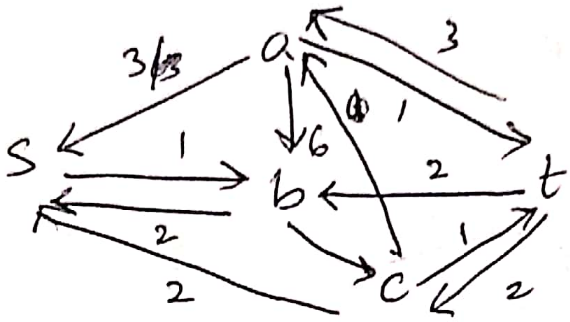
$\phi = 2$



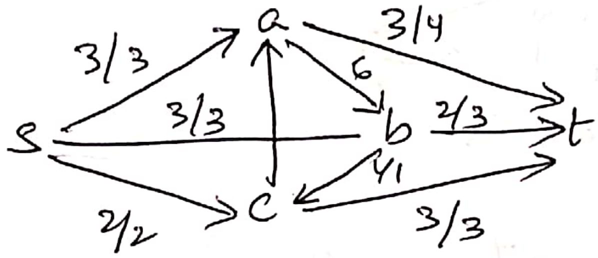


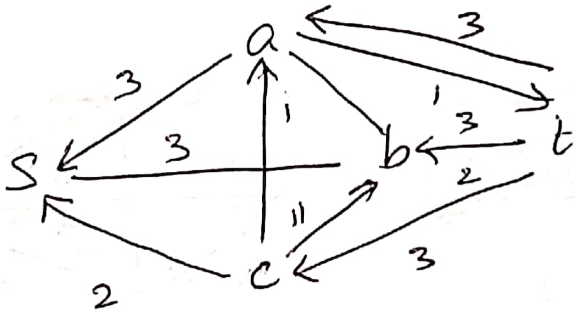
(iii) path : $s \rightarrow c \rightarrow t$ $\quad \phi = 2$





(iv) path : $s \rightarrow b \rightarrow c \rightarrow t$ $\quad \phi = 1$





max flow = sum of all flows = $3+2+2+1 = 8$

1) Given graph G. edge (u,v) of G

⇒ essential.

(a) To prove that G has essential edge.

Using max flow min cut theorem, we get that max flow of a graph is min cut. The sum of the capacities of some set of edges in the graph correspond to min cut.

Therefore, when the capacity is decreased in one of these edges, using max flow min cut theorem, max flow will also decrease

⇒ An essential edge is always present in graph G.

(b) Algorithm for essential edge in G:

* Consider graph G in which a set (u,v) exists in such a way that E an edge between them in G, let maxflow (a, b) = m and let residual graph be G_m

∴ Set of edges (u,v) has no path from u → v in ~~red~~ residual graph G_m. G_m is the essential edge.

* If there is no edge from $u \to v$, the capacity from $u \to v$ is reached, if we try to decrease the capacity by amount x while retaining the max flow value then we should find another route.

* But since there is no path from $u \to v$ it cant be rerouted

$\Rightarrow (u, v)$ is an essential edge.

* To find set of (u,v) edges where (u,v) is an edge in graph where there's no path in (u,v) in residual path.

* To find all strongly connected components of residual graph
  - represent of all DAG of all strongly connected
  - Topological sort on D_j to get
  - start with root of T_j for each node. we maintain set of vertices (u,v) is reachable from. Similarly find for all nodes.