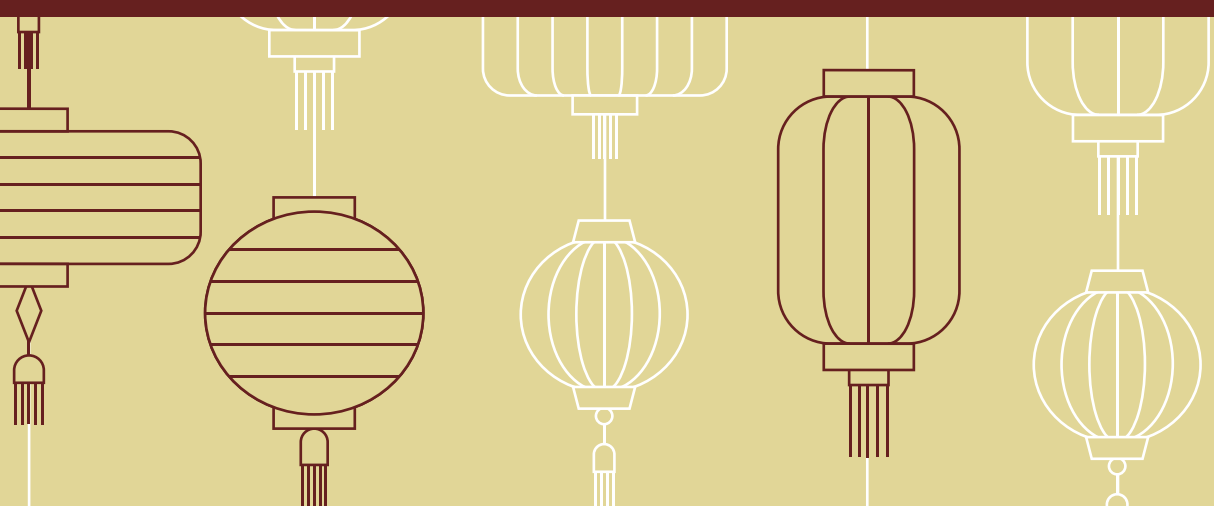
Shivambu Dev Pandey

Advanced Algorithm Design (CS366)



Q1.
- directed graph $G_1 = (V, E)$
- capacities c_e
- max flow f_e $\forall$ e
- source s
- sink t
- chosen edge $e^* = (u, v)$

We want to increase c_{e^*} by 1 and then find max flow for the new graph.

By Max flow - Min Cut Theorem,

E an s-t cut of cap. f_e in G_1

if we increase cap. of e^* by 1

then cut's cap. also increases by at most 1

$\Rightarrow$ $f_e \leq f_{e^*} \leq f_e + 1$

If cap. are integral,

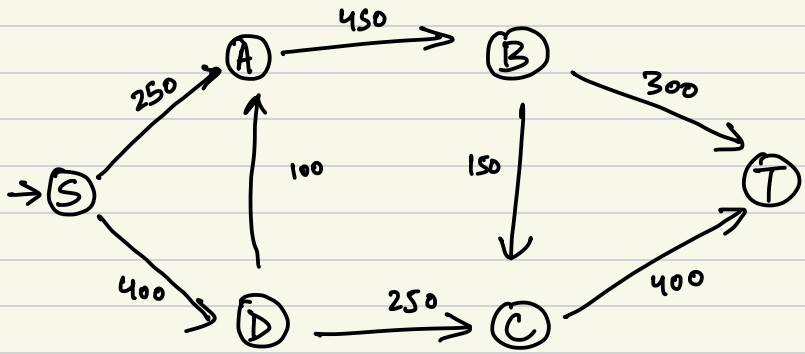
$f_{e^*} = f_e$ or $f_e + 1$

Algorithm :

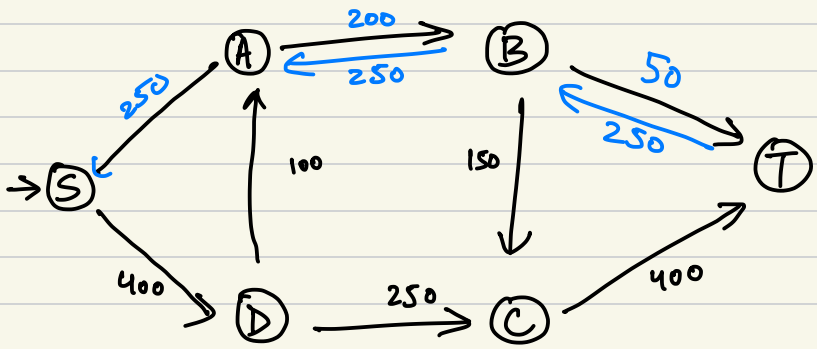
- Inc cap by 1
- Use the residual graph G_f' wrt the current flow f (only e^* forward residual changed)
- Run BFS from s on G_f' to find s-t path
- If no path found → return f
- Else recover the path,
  let Δ = min residual ($= 1$)
  foreach edge for. cap $-= \Delta$
  and rev. cap $+= \Delta$
  return augmented flow

Total $= O(1) + O(m+n) + O(n) + O(n)$
$= O(m+n)$

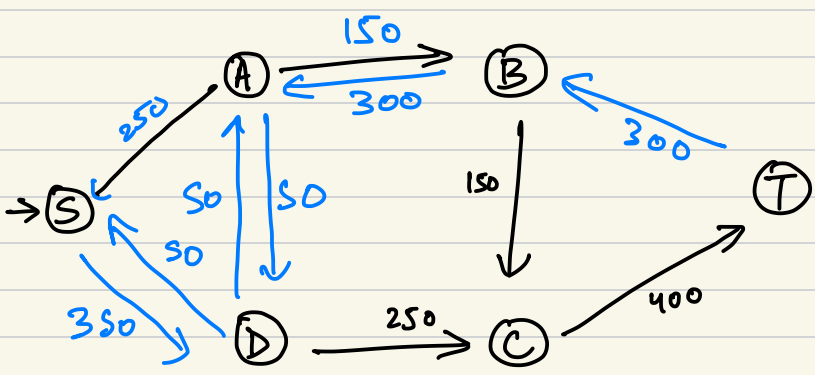
ف2.



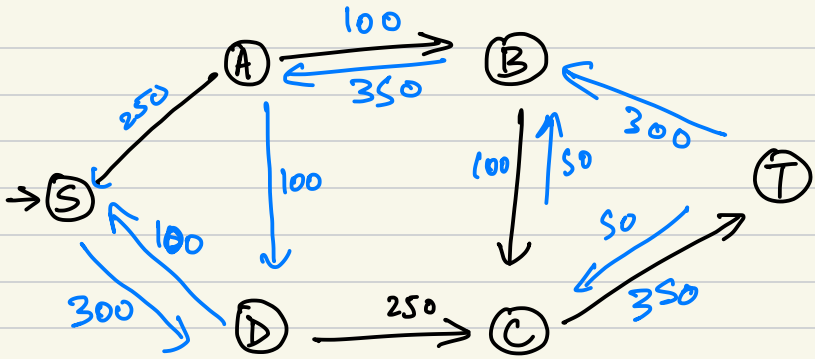
SABT , b = 250 , f = 250



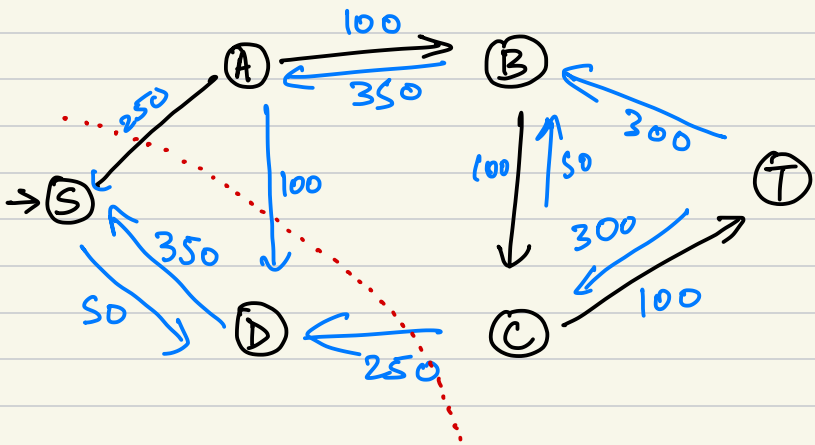
SDABT , b = 50 , f = 250+50 = 300



SDABCT , b = 50 , f = 350



SDCT , $b = 250$, $f = 600$



Min cut = {S, D} {A, B, C, T}

a) By above diagram, max flow = 600
Therefore maximum 600 vehicles can enter and leave the network every hour.

b) The cut edges S→A (250), D→A (100), D→C (250)
If we increase SA by 100 max flow = 700
DA max flow = 650
DC max flow = 650

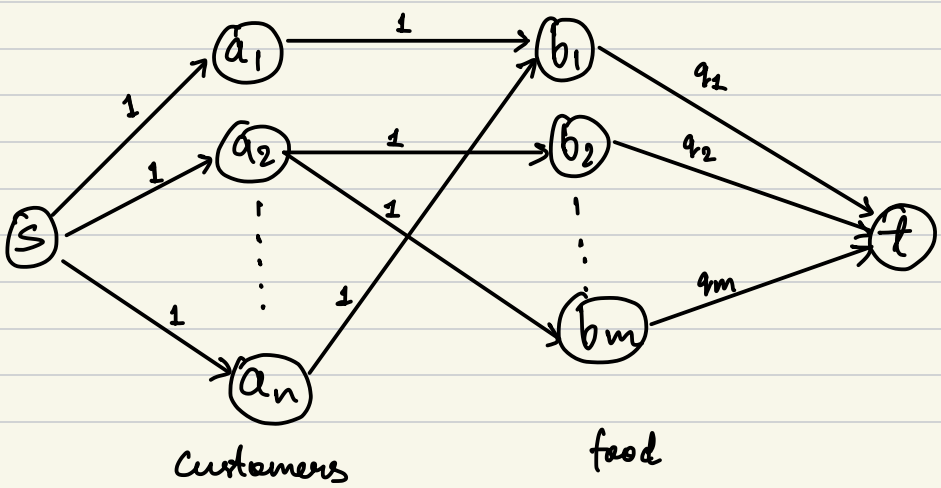
∴ $S \to A$ is the single section of the road that can be improved to increase the traffic flow

Q3.

For minimizing vouchers we need to satisfy maximum customers.

This can be done by finding the maximum bipartite matching.

We can create nodes for each customer and connect to source, connect each customer to food types, and food types to sink.



(s, a_i) edges with cap. = 1, for each $i: 1 \to n$

(a_i, b_j) edges with cap. = 1, for each $i: 1 \to n$, $j: 1 \to m$

(b_j, t) edges with cap. = q_i, for each $j: 1 \to m$

After building the network,

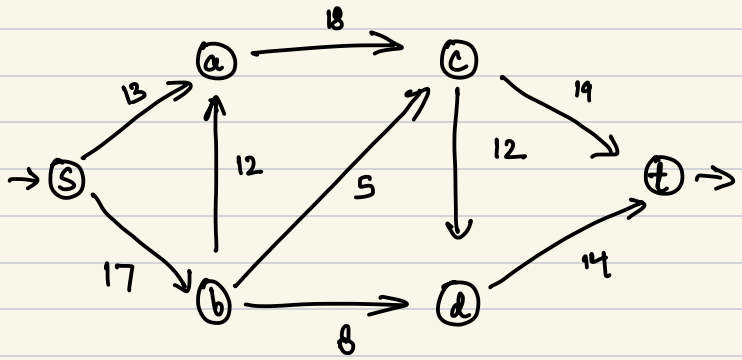
- Run a max-flow algorithm like Ford-Fulkerson to compute max s-t flow f.

- Now, for each edge (a_i, b_j) with $f(a_i, b_j) = 1$ assign food b_j to customer a_i.

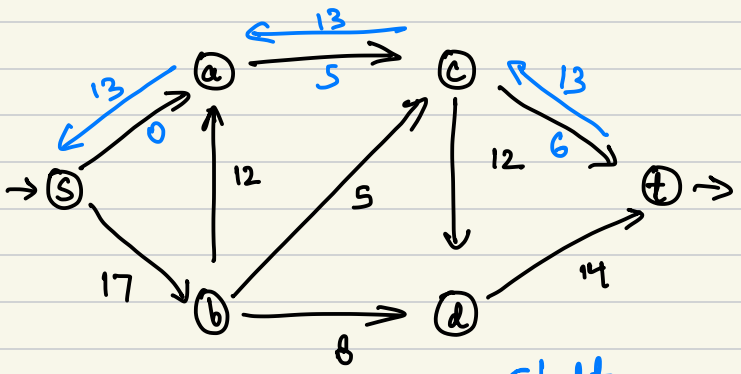
- for each customer a_i with no outgoing unit flow, give Rs. 100 voucher

Number of vouchers given is $n - f$, where f is the max flow.

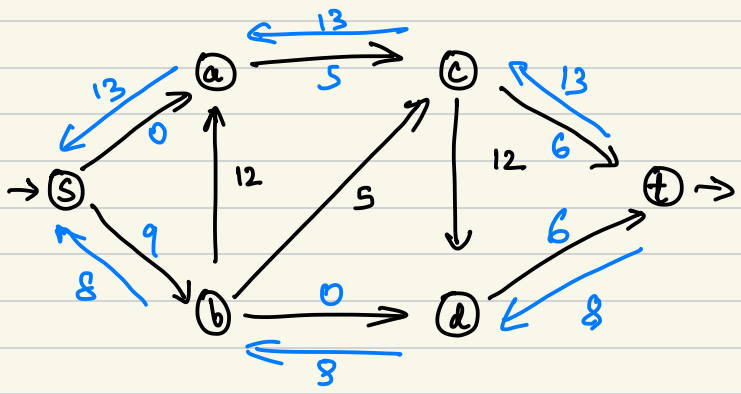
Q4.



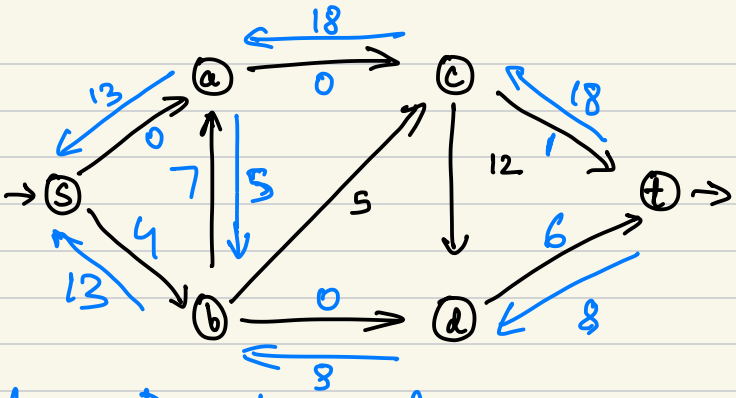
sact , B = 13 , f = 13



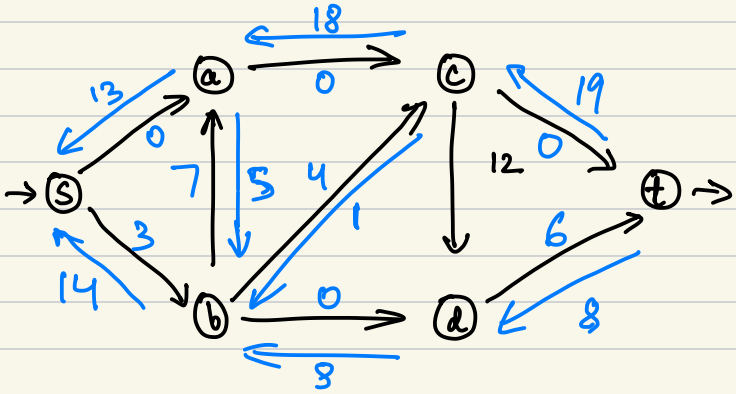
sbdt , B = 8 , f = 21



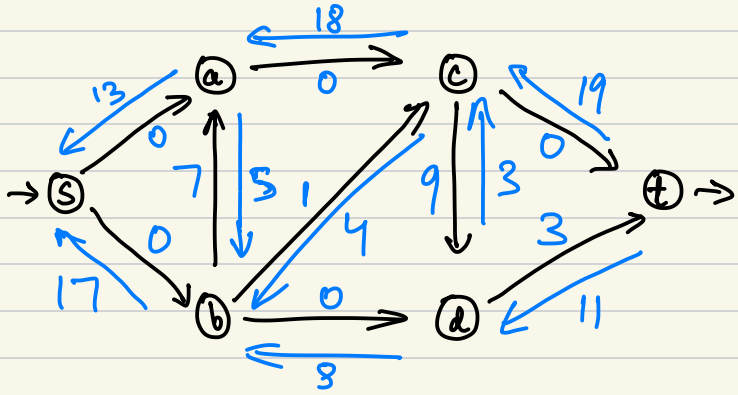
sbact , B = 5 , f = 26



sbct , B = 1 , f = 27



sbcdt , B = 3 , f = 30



∴ The max flow is 30

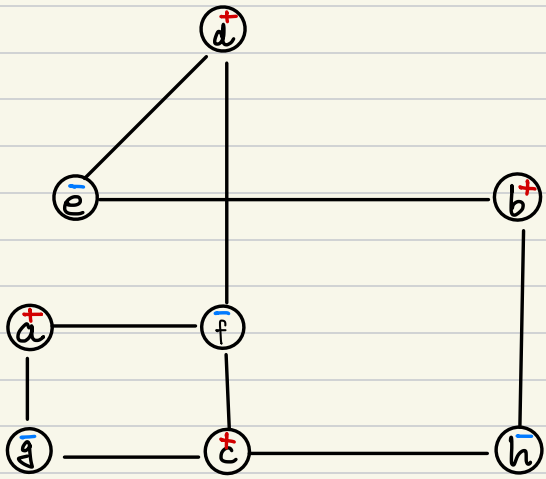
Q5.

- Compute a minimum s-t cut C and its value $\lambda = |C|$
- let the edges in C be $E(C) = \{e_1, e_2 \cdots, e_k\}$
- for each $e_i \in E(C)$
  - Increase cap of e_i by 1
  - recompute min cut in new graph, val $= \lambda_i$
  - restore original cap. of e_i
- If for any i we find $\lambda_i = \lambda$, **min cut is not unique**

  because after increasing e_i, there exists a cut C_i of cap. λ.
  Since we only inc. e_i by 1, that cut C_i already had cap $\leq \lambda$ in original graph, hence it had capacity exactly λ there as well.
  Because $C_i \neq C$, cut is not unique
- If for every i we get $\lambda_i > \lambda$, **min cut is unique**

  because if there is a different min cut $C' \neq C$ of value λ, then some edge $e_i \in C$ is not in C'.
  $\therefore$ inc. the cap. of that particular e_i does not affect C' so the global minimum stays λ.
  Therefore some $\lambda_i = \lambda$ will be observed if non-uniqueness is detected.

Q6.



a) Yes G_1 is bipartite

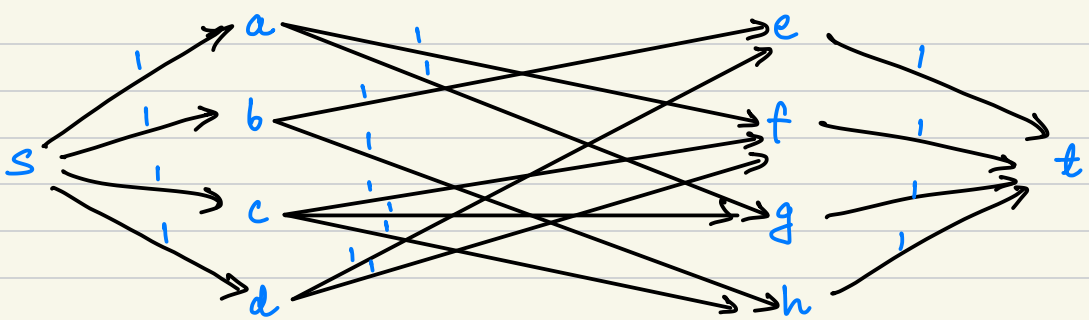
on 2-colouring, $L = \{a, b, c, d\}$
$R = \{e, f, g, h\}$

Maximum matching, $M = \{(a, g), (c, f), (b, h), (d, e)\}$

$|M| = 4$

b)
- A perfect matching in a graph is a matching that covers every vertex of the graph. It has size $|V|/2$ when $|V| \% 2 == 0$
  Example (a)
- In a there is perfect matching.
  But having a maximum matching does not automatically mean it is perfect unless all vertices are covered.

c)

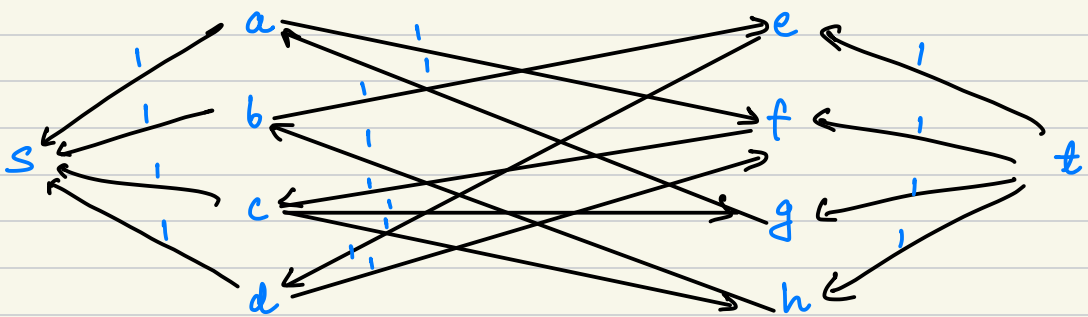


sagt , B=1 , f = 1

sbht , B=1 , f = 2

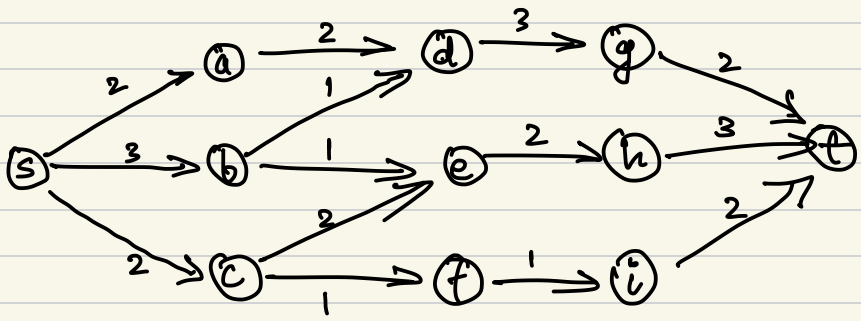
scft , B=1 , f= 3

sdet , B=1 , f= 4

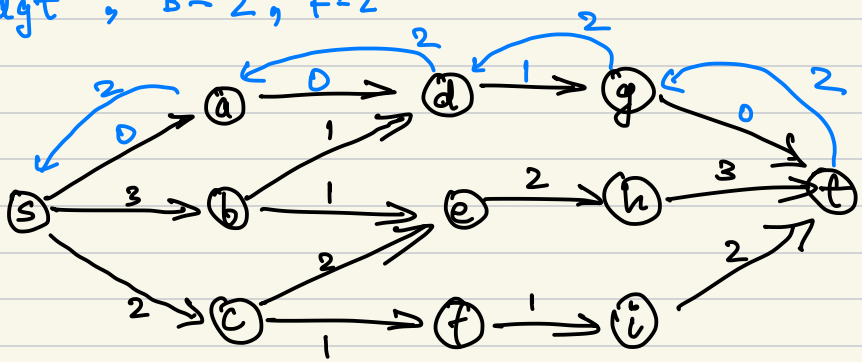


∴ |f| = 4 = |M|

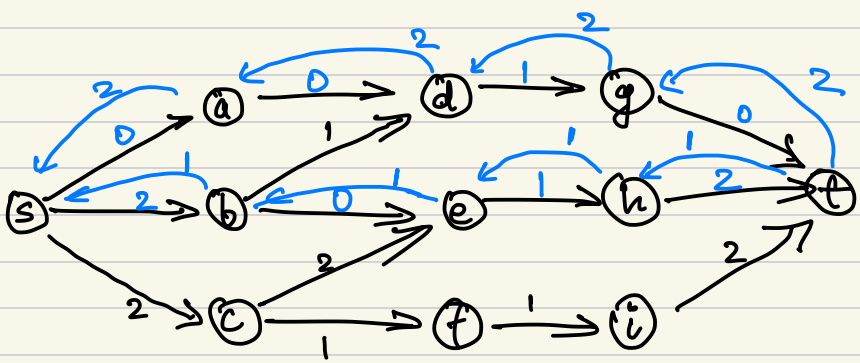
q7.



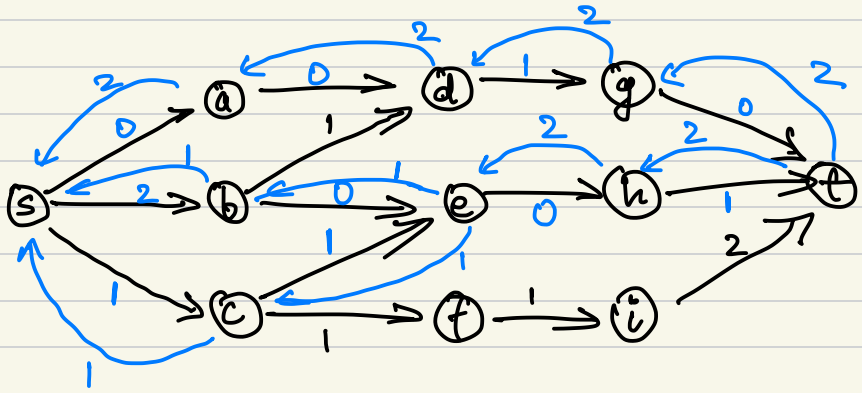
sadgt , B=2, f=2



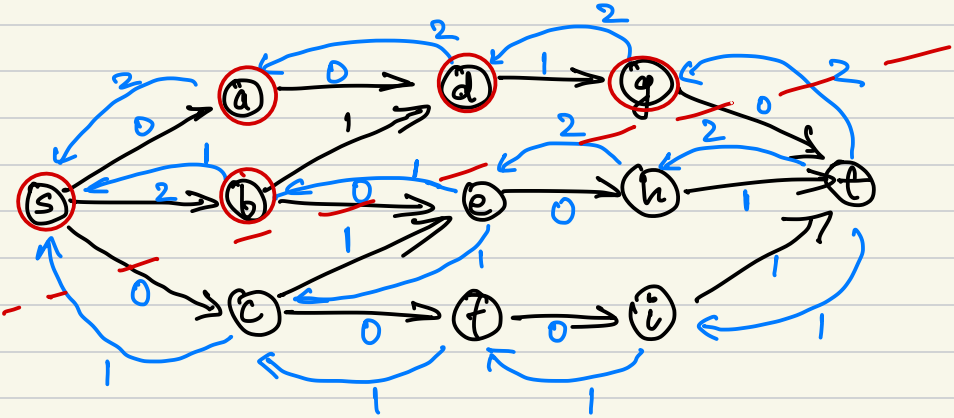
sbeht , B=1, f=2+1



sceht , B=1 , f= 2+1+1



Scfit , B=1 , f = 2+1+1+1



Min cut is {s, a, b, d, g} {c, e, f, h, i, t}

Min cut capacity

c) Ford-Fulkerson keeps augmenting along s-t paths in the residual graph until no s-t path exists.
At that point the current flow is maximum.

Complexity :

Augmenting Path : BFS/DFS → O(V+E)

here $E \geq V-1$

$\therefore O(E)$

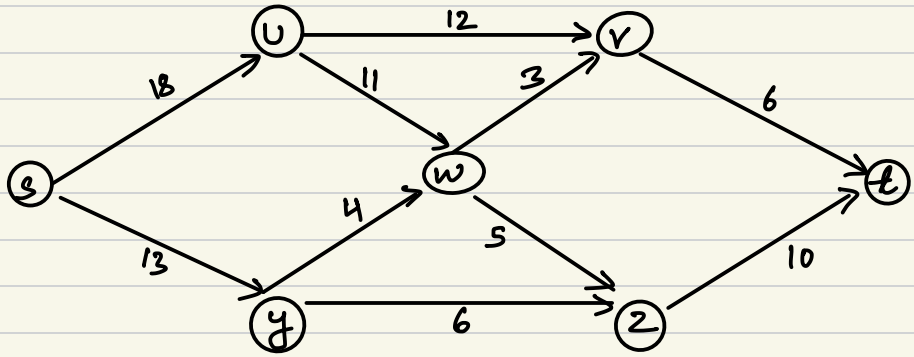
Iterations : Each aug. inc. flow ≥ 1 units

$\therefore O(F)$

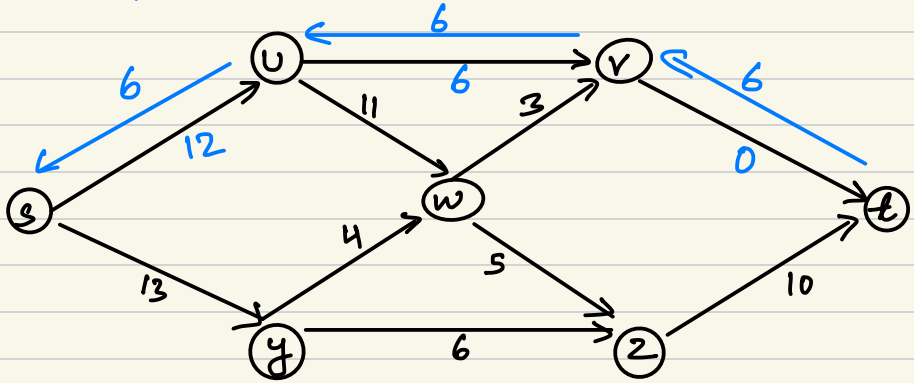
Time complexity = $O(F \cdot E)$

It is pseudopolynomial as it depends on maxflow f

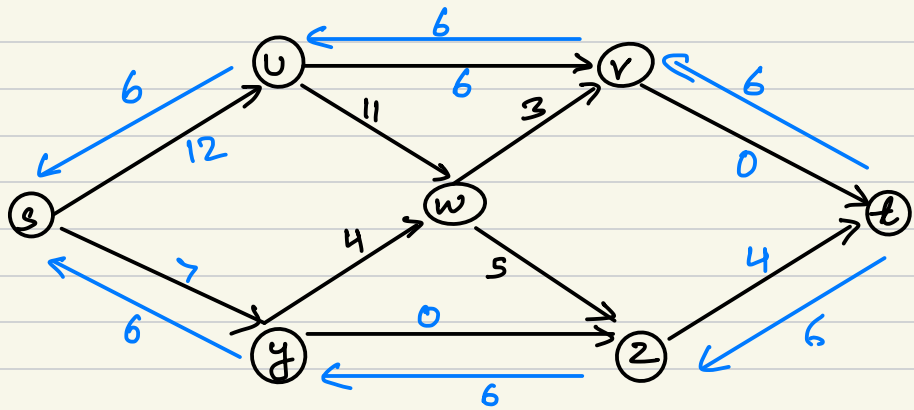
Q8.



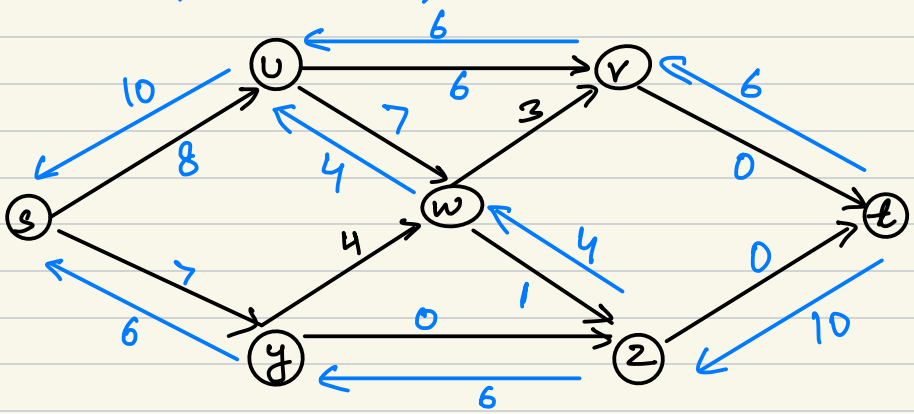
sut , B = 6 , f = 6



syzt , B = 6 , f = 12

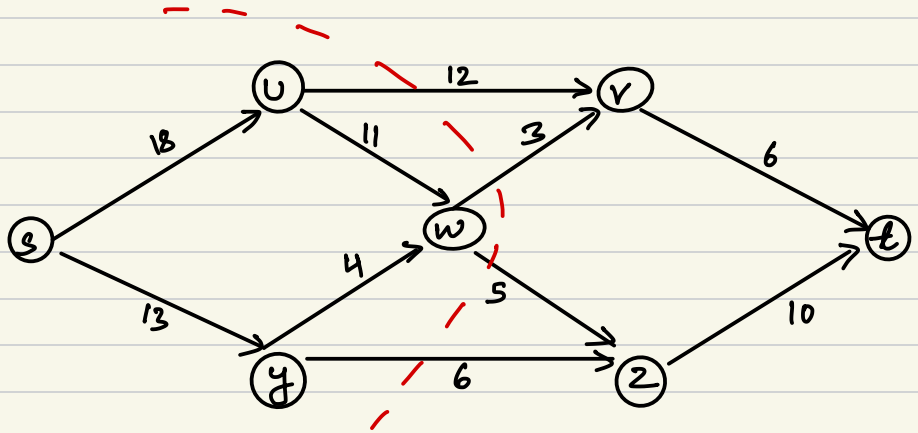


suwzt , B = 4 , f = 16



c. Max flow = 16

d.



A = {s, u, y, w} , B = {v, z, t}

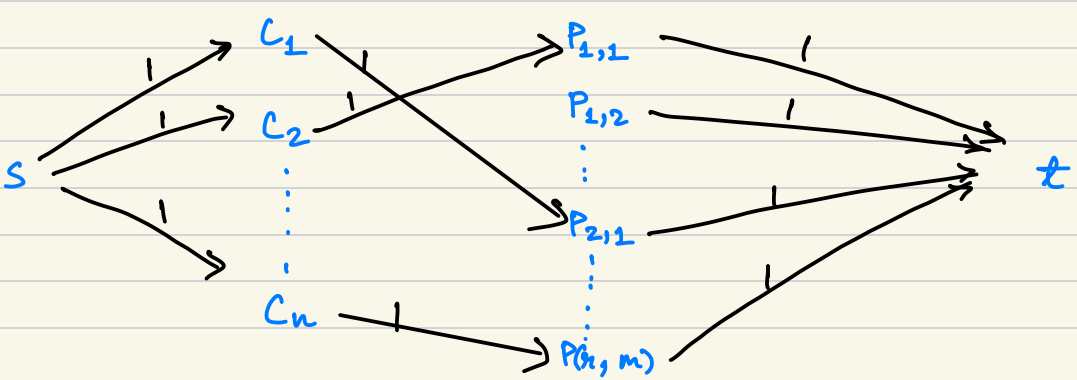
cap of C(A,B) = 12 + 3 + 5 + 6
= 26

Q 9.

We can solve this problem by reducing to a max-flow / bipartite matching

Let us build $G = (V, E)$

- source : s , sink : t
- vertex C_i for each course $i : 1..n$
- vertex $P_{(j,k)}$ for each room $j : 1..r$ & timeslot $k : 1..m$
- Adding edges
  - $s \to C_i$ with cap 1 for every course i
  - $C_i \to P_{(j,k)}$ with cap 1 iff $E[i] \subseteq S[j]$
  - $P_{(j,k)} \to t$ with cap 1 for every room-timeslot



This is a bipartite network.

We can run max-flow algorithm to find the maximum bipartite matching.

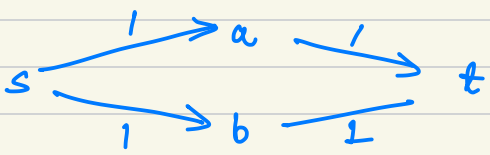
feasible schedule means $f = n$.

10.

a) True
Every cut capacity is even, therefore min cut is even, hence, max flow is even

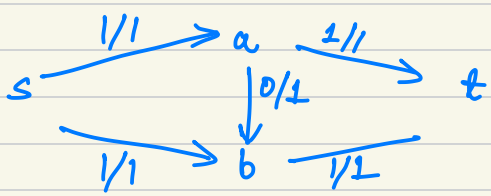
b) True
Every edge is even, divide capacities by 2. Then find integer max flow and double it.

c) false



Every edge is odd but max flow is even

d) false



0 is not odd.

Q11.

$$|c| + \min(0, \Delta) \leq |c'| \leq |c| + \max(0, \Delta)$$

$$\therefore \quad |c' - c| \leq |\Delta|$$

a) True
if $\Delta = +1$, $|c' - c| \leq 1$

b) True
if $\Delta = +k$, $|c' - c| \leq k$

c) True
if $\Delta = -1$, $|c' - c| \leq 1$

d) True
if $\Delta = -k$, $|c' - c| \leq k$

Q12.

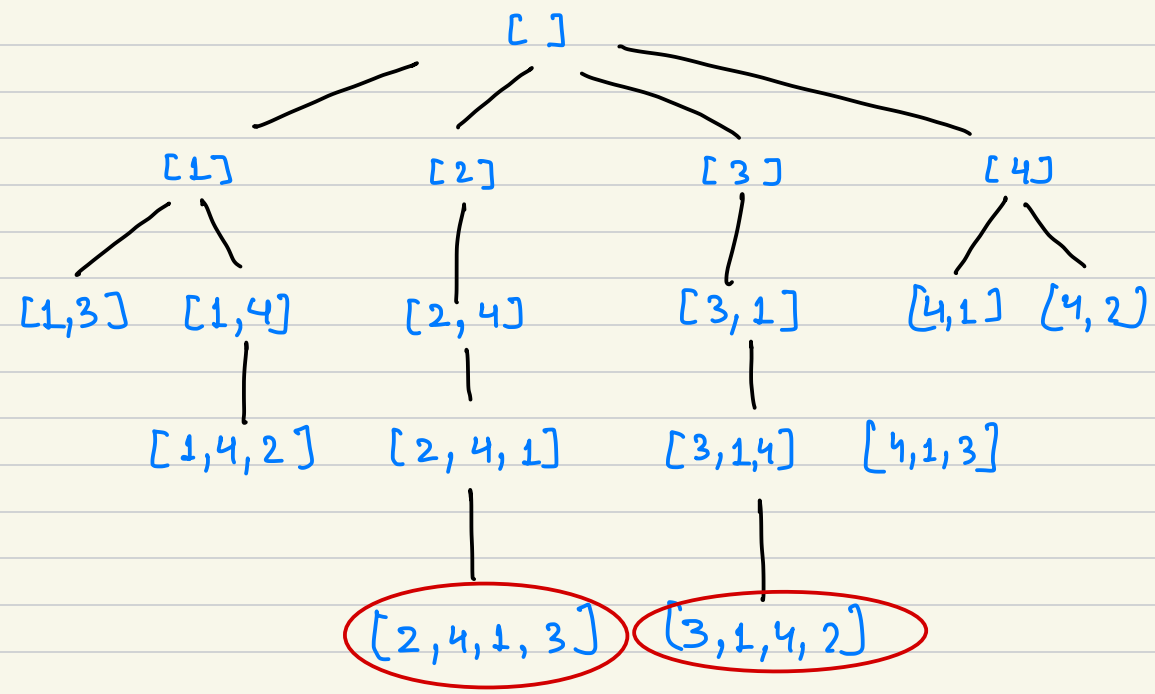
```
void construct_candidates ( int k, int S[], int K,
                            int *nc, int curr_sum, int rem_sum ){

    *nc = 0

    if ( curr_sum + S[k] <= K ) {
            c[*nc] = 1 ;
            *nc    = *nc + 1 ;
    }

    if ( curr_sum + ( rem_sum - S[k] ) >= K ){
            c[*nc] = 0 ;
            *nc = *nc + 1 ;
    }

}
```

Q13.

$[c_1, c_2, c_3, c_4]$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$q_1 \quad q_2 \quad q_3 \quad q_4$



Number of solutions = 2

4 backtracks before we get first solution

b) No diagonal constraint then it becomes permutation backtracking.

level k has $P(N,k) = \frac{N!}{(N-k)!}$

Total nodes $= \sum_{k=0}^{N} P(N,k) = \sum_{k=0}^{N} \frac{N!}{(N-k)!}$

c) Number of complete assigment $= N^N$

```
vector<int> construct_candidates (int k, Put chosen,
                                  Put n, Ink K)
{
    vector <Dnt> c ;
    Int rem = n - k + 1 ;

    if C Chosen < K 44 chosen + remaining >= k)
        C. push_back (1);

    if (chosen + C remaining - 1) >= k)
        C. push_back (0);

    return c ;
}
```