HU, Wenbin 20760747 whuak@connect.ust.hk.

Problem 1:

We need to prove: for a connect $G_1 = (V, E)$ with distinct weights and unique MST: $T = (V, E')$

If $e \in E'$, then there exists some subset of vertices $S_e \subset V$ such that:

$$w(e) = \min\{w(e') : e' = (u', v') \in E,\ u' \in S_e,\ v' \notin S_e\}.$$

proof: suppose: if $e \in E'$, then there dosen't exist any subset of vertices $S_e \subset V$ suchthat:

$$w(e) = \min\{w(e') : e' = (u', v') \in E,\ u' \in S_e,\ v' \notin S_e\}.$$

that mean. $\forall S_e \subset V$, $w(e) > \min\{w(e') : e' = (u', v') \in E,\ u' \in S_e,\ v' \notin S_e\}$

let $e \in (u, v)$, $u \in S_e$, $v \notin S_e$

let $e_m = \min\{w(e') : e' = (u', v') \in E,\ u' \in S_e,\ v' \notin S_e\}$

since T is a tree, so e is the only edge that connects S_e and $V - S_e$

Adding e_m to T creates a cycle that contain both e_m and e. removing any edge from the cycle will keep the graph a tree.

$\therefore$ $T \cup \{e_m\} - \{e\}$ is a tree, denote it as T^*

$\therefore w(T^*) = w(T \cup \{e_m\} - \{e\}) = w(T) + w(e_m) - w(e)$

since $w(e_m) = \min\{w(e') : e' = (u', v') \in E, u' \in S_e, v' \notin S_e\} < w(e)$ (since every edge in E is distinct)

$\therefore w(e_m) - w(e) < 0$

$\therefore w(T^*) = w(T) + \left(w(e_m) - w(e)\right) < w(T)$

$\Rightarrow$ Contradicting fact that T is a Minimum spanning tree

$\therefore$ the statement is True

$\therefore$ proved!

Problem 2:

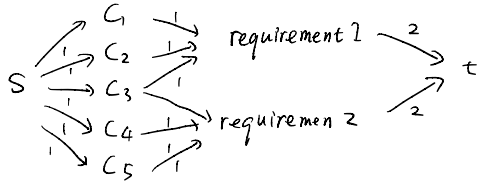
(A)

Algorithm ($X_{i,j}$, N_i, L_j):

```
1   total ← 0                                          // total is total number of course must be taken for requirement.
2   for i←1 to m:
3       total ← total + N_i[i]
4   create a Graph (V,E) with m vertices denote as V_Ri   // m vertice is for m requirements.
5   for j←1 to n:                                      // create bipartite matching between courses taken and requirements.
6       if L_j[j] = 1 :
7           add a vertex V_Cj to G
8           for i←1 to m:
9               if X_i,j[i,j] = 1                      //create edges if course taken, connect it to fulfilled requirements.
10                  Create an edge bewteen V_Cj and V_Ri,
11                  direct the edge from V_Cj and V_Ri, weight it as 1
12  create edge s, connect it to all existed courses V_Cj, direct the edge from s to V_Cj, weight 1
13  Create an edge t.
14  for i←1 to m:
15          connect V_Ri to t. direct the edge from V_Ri to t, weight N_i[i]
16  max_flow ← Ford-Fulkerson (G, s, t)                // get the max flow from the subroutine Ford-Fulkerson we learnt from class.
17  if max_flow = total                                // if max flow is greater than total number of courses of the requirement, then the student can graduate, return true
18      return True
19  else
20      return false
```

explanation: take example in the question:

we create a graph: (if the student take C_1, C_2, C_3, C_4, C_5)



Max flow = 4, which is the total number of courses of requirement, so the student can graduate.

(B) line 1~4 : $O(m)$

line 5~11 : $O(mn)$

line 12~15 : $O(m+n)$

line 16 : we analyze Fold-Fulkerson algorithm :

running time = $O(|f^*| \cdot E)$, $|f^*|$ is the number of iterations, E is the number of edges.

$|f^*| \leq \max\{m, n\}$.

but number of requirements must less than or equal to number of total course.

$\therefore m \leq n$

$\therefore |f^*| \leq n$.

the graph have at most $mn+m+n$ edges.

$\therefore |E| \leq mn+m+n$

$\therefore$ running time of Ford-Fulkerson $= O(|f^*| \cdot E) = O(n \cdot (mn+m+n)) = O(n^2 m)$

other lines : $O(1)$

$\therefore$ in total : running time $= O(m) + O(mn) + O(m+n) + O(mn^2)$

$= O(n^2 m)$

(C)

firstly, we get the total number of courses for requirements, by doing a for loop for N:[1...m] and adding them together.

We create a graph, whose nodes includes courses taken by the student and total requirements

we connect courses to requirements which they can fulfill respectively, direct those course nodes and their corresponding requirements. and weight the edge as 1.

so, now we model it as a Bipartite Matching Problem.

add an node s to the graph and connect it to all course nodes. weighted 1, direct s to courses

add another node t to the graph and connect it to all requirement nodes. weighted N:[i] direct requirements to t.

Then we do Folk-Fulkerson Algorithm to get max flow

Magtitude of the max flow represents the total number courses can fulfill requirements without overlap.

Check if the max flow is equal to total courses of requirements. if so, the student can graduate otherwise can't.

$\therefore$ the algorithm can always find whether the student can fulfill all requirement.

$\therefore$ the algorithm is correct.

Problem 3:

(a)

(i) Algorithm:

weight every edge in G as 1. do Ford-Fulkerson(G, s, t) algorithm taught in Fax-Flow.

Find max flow, and find flow in every edge.

Do BFS from s in G_f. when it stops, we put those vertices visited by BFS in S

put the rest in T. get the cut(S,T), and it is a Min-Cut in G

then output the set of edges in the Min-Cut, which is a smallest seperating edge set of G.

(ii) proof for its correctness:

Claim: Let S-T be any s-t cut. Then the set of edges crossing between S-T is a seperating edge set.

proof for then claim:

def of seperating edge set: A subset $E' \subseteq E$ s.t. if after removing E' from G, no s-t path exists.

def of s-t cut: a partition (S,T) of V with $s \in S$, $t \in T$.

edges crossing between S-T all exactly have one end in S. and another end in T

so these edges connects S to T.

if we remove these edges, then there will be no connection between S-T.

since $s \in S$. $t \in T$, there will be no path from S to T.

so the set of edges crossing between S-T is a seperating edge set.

$\therefore$ the claim is proved.

Solving the disjoint s-t path problem use the Max-flow algorithm (in class: Application of Max flow).

so in my algorithm, those flows in path from s to t implies the disjoint s-t pathes in G, since all edges has capacity 1.

we can do BFS (taught in Basic Graph Algorithms) from s (in G_f) and find the Min-Cut (in G) and its set of edges.

In class Max Flow we know:

① $\exists (S,T)$, s.t. max-flow $|f| = C(S,T)$ (from Max flow Min-Cut Thm)

② Claim: for any flow f and any cut: $|f| \leq C(S,T)$

from ①②: max flow $|f| = |$Min-Cut$|$

$\because$ subset edges E_1 in the Min-Cut is a seperating edge set. (from the claim we proved just now)

$\therefore |E_1|$ is greater or equal to the size of smallest seperating edge set

, that is: max flow $|f| \geq$ size of smallest seperating edge set (*)

from ②: max flow $|f| \leq |$any cut $C(S,T)|$

$\therefore$ max flow $|f| \leq$ size of any seperating edge set (from the claim we proved just now)

$\therefore$ max flow $|f| \leq$ size of smallest seperating edge set. (**)

from (*), (**) : max flow |f| = size of smallest seperating edge set = |Min-Cut| ..... (1)

from the claim : Min-Cut is a seperating edge set ..... (2)

∴ Min-Cut return by my algorithm is exactly a smallest seperating edge set (from (1), (2))

∴ proved !

* supplymentary proof for : we can do BFS algorithm in G_f to find the min-cut

proof: BFS can't go from S to T ⇒ there is no path from S to T

which means : $\forall e \in (S,T)$ is full i.e. $f(e) = c(e)$. $\forall e' \in (T,S)$ is empty i.e. $f(e') = 0$

∴ (S,T) is a Min-Cut ∴ proved.

(iii) running time for Ford-Fulkerson Algorithm

$= O(|f| \cdot |E|) = O(|E|^2)$

(since every edge weight 1, max flow $|f| \leq$ |edges connected to s| $\leq |E|$)

running time for BFS Algorithm.

$= O(|V| + 2|E|) = O(E)$ (by the assumption : $V = O(E)$)

∴ total running time is : $O(|E|^2)$

(b) from the Question :

- For (b), modify G to create a new graph $G' = (V', E')$.
  - (i) For every vertex $v \in V$, create two vertices v_ℓ, v_r in V'.
  - (ii) For every vertex $v \in V - \{s, t\}$, Create edge (v_ℓ, v_r) in E'.
  - (iii) For every edge $(u, w) \in E$, create edge (u_r, w_ℓ) in E'.
  - (iv) Remove vertices s_ℓ and t_r from V'.

  Note that if P is an $s - t$ path in G that passes through vertex v, the corresponding path P' in G' is an $s_r - t_\ell$ path that passes through edge (v_ℓ, v_r).
- The modified graph G' contains two very different type of edges:

$$E'_1 = \{(v_\ell, v_r) : v \in V - \{s, t\}\},$$
$$E'_2 = \{(u_r, v_\ell) : (u, v) \in E\}.$$

  A separating vertex set in G corresponds to a separating edge set in G' in which all edges are in E'_1 and vice-versa.

  So solving (b) corresponds to finding a smallest separating edge set in which all edges are in E'_1.

  The idea is to appropriately modify the algorithm for part (a) to find such a set.
- Suppose you assign capacity 1 to all edges in E'_1 and capacity 2 to all edges in E'_2. Prove that the edges crossing a Min-Cut must all be of type E'_1.

  (Proving this requires understanding the structure of the min-cut in the proof of the correctness of the Ford-Fulkerson algorithm.)
- Use the observations above and (a modified version of) the result of part (a) to solve (b).

(i) Algorithm:

we assign capacity 1 to all edges in E_1' and capacity 2 to all edges in E_2'

we do Fork-Fulkerson Algorithm (G', s_r, t_l), get the max flow.

and we do BFS Algorithm for the residual graph, created by Ford-Fulkerson algorithm. (from s_r)

when BFS stops, we put those vertices in G' (whose corresponding vertices in G_f' visited by BFS) in vertices subset S_r, and put rest vertices in G' in T_L

the s_r-t_l cut : (S_r, T_l) is a Min-Cut in G'

return corresponding vertices(in G) of every edges (V_l, V_r) in the Min-Cut.

(ii) proof its correctness:

from (a) we know: we can find a smallest seperating edge set by doing Ford-Fulkerson algorithm, BFS algorithm to find the Min-Cut in G.

so in (b): the Min-Cut we find by the algorithm is a smallest seperating edge set in G'

now, we prove (claim1):

The edges crossing a Min-Cut must all be of type E_1'

proof of the claim1:

from (a)'s proof, we know we can get a Min-Cut in G', let it be (S', T')

and the Min-Cut has these properties: (from Max-flow Min-Cut Thm)

all edges e from S' to T' are full, $f(e) = c(e)$

all edges e from T' to S' are empty, $f(e) = 0$

in (b), we know capacity of vertices in G' is 1 or 2.

so the flow in every path from s_r to t_l is 1.

∴ in G_f': for edges in E_1': residual capacity for (V_e, V_r) is 0, flow is 1

for edges in E_2': residual capacity for (V_r, V_l) is 1, flow is 1

suppose there is some edge $e \in E_2'$ in the Min-Cut (S', T')

so there is some residual capacity from S' to T', which means there will be extra flows from S' to T', which contradict the properties of Min-Cut's property.

∴ there will be any edge $e \in E_2'$ in Min-Cut.

∴ The edges crossing a Min-Cut must be of type E_1'

∴ Claim 1 is proved.

since A separating vertex set in G corresponds to a seperating edge set in G' in which all edges are in E', vice-versa.

we proved that the Min-Cut we find by the algorithm corresponds a smallest seperating vertices set.

∴ correctness of the algorithm is proved.

(iii) running time of Ford-Fulkerson Algorithm

$= O(|f|(|V|+|E|-2)) = O((|V|+|E|-2)^2) = O((|V|+|E|)^2) = O(2|V| \cdot 2|E|) = O(|V||E|)$

Assume: $|V| = O(|E|)$

- $|f|$: max flow
- $|V|+|E|-2$: # of Edges in G'
- $|f| \le |V|+|E|-2$

Running time of BST Algorithm

$= O((2|V|-2)+2(|V|+|E|-2)) = O(4|V|+3|E|) = O(|E|)$

Assume: $|V| = O(|E|)$

G'_f has $2|V|-2$ vertices and $2(|V|+|E|-2)$ edges.

∴ in Total: running time $= O(|V||E|)$

Problem 4.

(a)

$$D^{(2)} = \begin{pmatrix} 0 & 5 & 10 & 11 & 3 & 5 \\ 6 & 0 & 16 & 3 & 5 & 13 \\ \infty & 3 & 0 & \infty & 1 & 3 \\ \infty & 4 & 15 & 0 & 2 & 4 \\ 8 & 2 & 7 & 3 & 0 & 2 \\ \infty & \infty & 5 & 1 & 3 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 5 & 10 & 6 & 3 & 5 \\ 6 & 0 & 12 & 3 & 5 & 7 \\ 9 & 3 & 0 & 4 & 1 & 3 \\ 10 & 4 & 9 & 0 & 2 & 4 \\ 8 & 2 & 7 & 3 & 0 & 2 \\ 11 & 5 & 5 & 1 & 3 & 0 \end{pmatrix} \quad D^{(8)} = \begin{pmatrix} 0 & 5 & 10 & 6 & 3 & 5 \\ 6 & 0 & 12 & 3 & 5 & 7 \\ 9 & 3 & 0 & 4 & 1 & 3 \\ 10 & 4 & 9 & 0 & 2 & 4 \\ 8 & 2 & 7 & 3 & 0 & 2 \\ 11 & 5 & 5 & 1 & 3 & 0 \end{pmatrix}$$

(b)

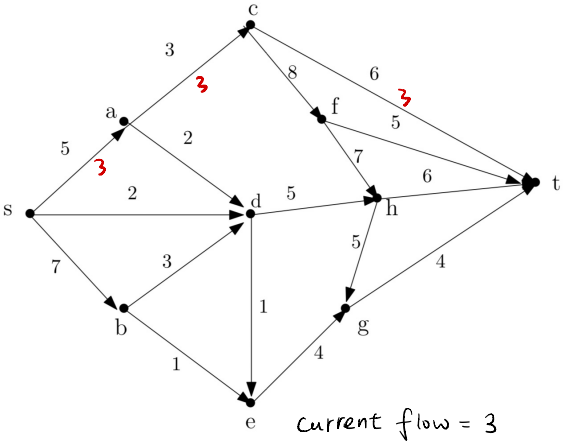
$$D^{(1)} = \begin{pmatrix} 0 & 8 & 10 & \infty & 3 & \infty \\ 6 & 0 & 16 & 3 & 9 & \infty \\ \infty & \infty & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 & 10 \\ 8 & 2 & 18 & 5 & 0 & 2 \\ \infty & \infty & 5 & 1 & \infty & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & 8 & 10 & 11 & 3 & \infty \\ 6 & 0 & 16 & 3 & 9 & \infty \\ \infty & \infty & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 & 10 \\ 8 & 2 & 18 & 5 & 0 & 2 \\ \infty & \infty & 5 & 1 & \infty & 0 \end{pmatrix} \quad D^{(3)} = \begin{pmatrix} 0 & 8 & 10 & 11 & 3 & \infty \\ 6 & 0 & 16 & 3 & 9 & \infty \\ \infty & \infty & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 & 10 \\ 8 & 2 & 18 & 5 & 0 & 2 \\ \infty & \infty & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 8 & 10 & 11 & 3 & 21 \\ 6 & 0 & 16 & 3 & 5 & 13 \\ \infty & \infty & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 & 10 \\ 8 & 2 & 18 & 5 & 0 & 2 \\ \infty & \infty & 5 & 1 & 3 & 0 \end{pmatrix} \quad D^{(5)} = \begin{pmatrix} 0 & 5 & 10 & 8 & 3 & 5 \\ 6 & 0 & 16 & 3 & 5 & 7 \\ 9 & 3 & 0 & 6 & 1 & 3 \\ 10 & 4 & 20 & 0 & 2 & 4 \\ 8 & 2 & 18 & 5 & 0 & 2 \\ 11 & 5 & 5 & 1 & 3 & 0 \end{pmatrix} \quad D^{(6)} = \begin{pmatrix} 0 & 5 & 10 & 6 & 3 & 5 \\ 6 & 0 & 12 & 3 & 5 & 7 \\ 9 & 3 & 0 & 4 & 1 & 3 \\ 10 & 4 & 9 & 0 & 2 & 4 \\ 8 & 2 & 7 & 3 & 0 & 2 \\ 11 & 5 & 5 & 1 & 3 & 0 \end{pmatrix}$$

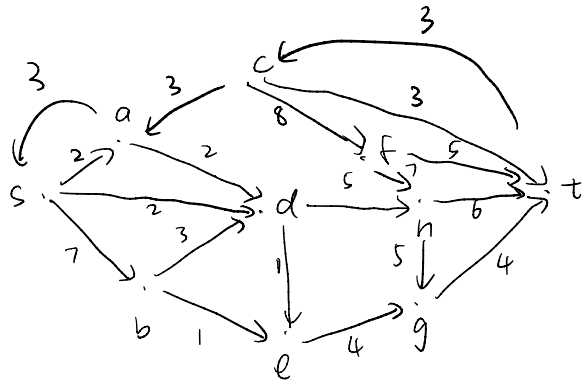
Problem 5:

(a)

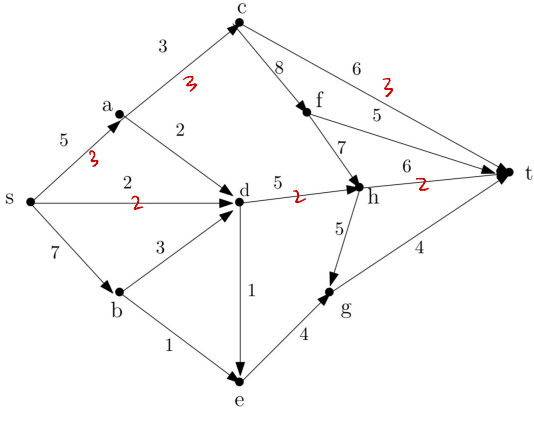
Step 1:



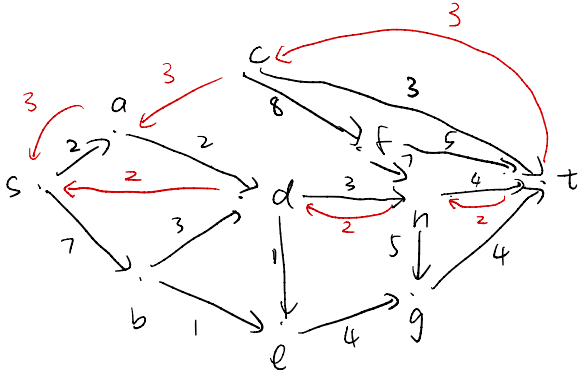
current flow = 3



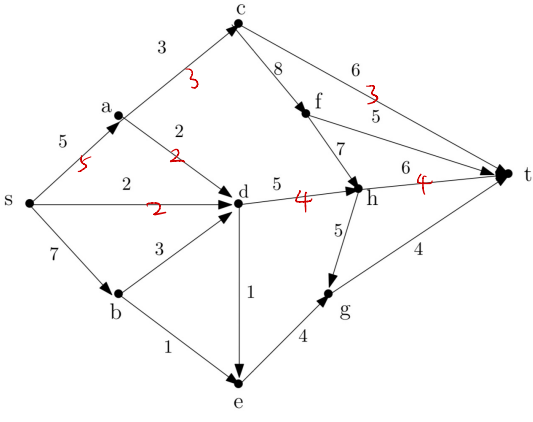
Step2:

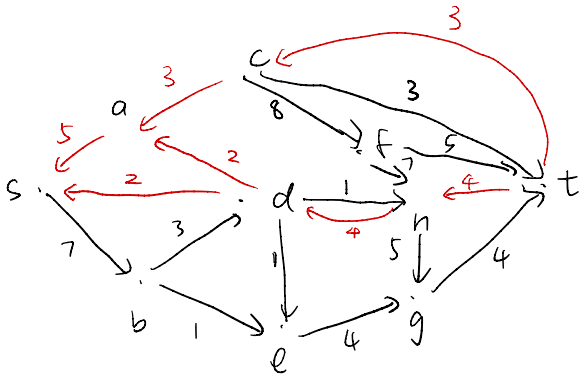


current flow = 5



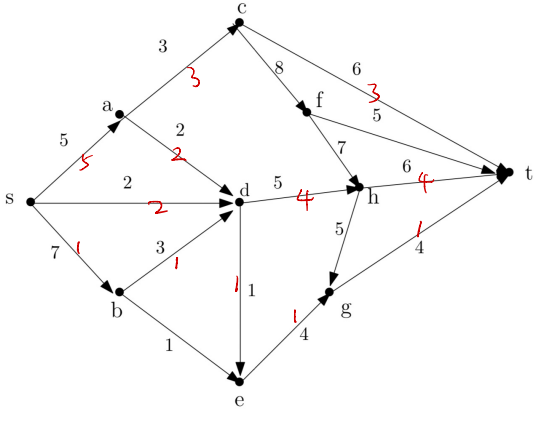
Step 3:

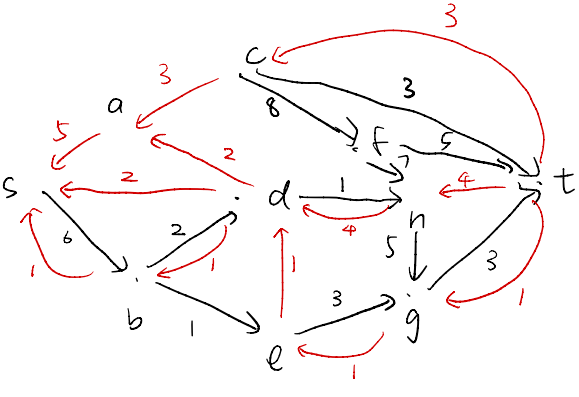




Current flow : 7.

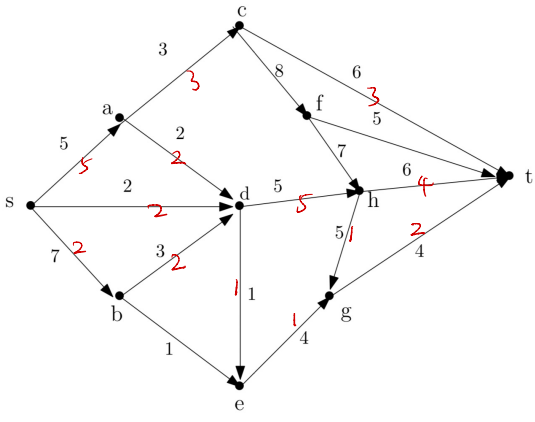
Step 4:



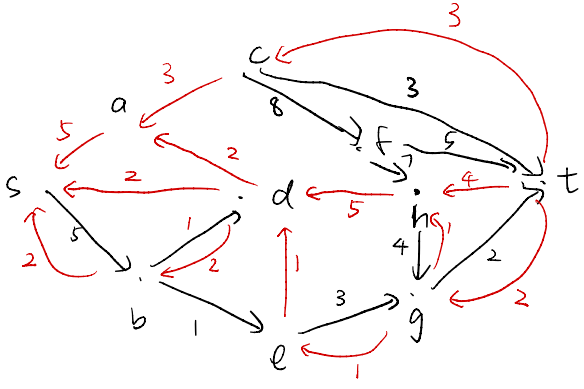


Current flow : 8

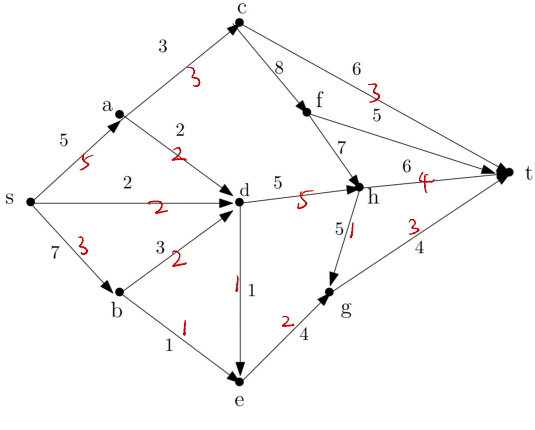
Step 5:



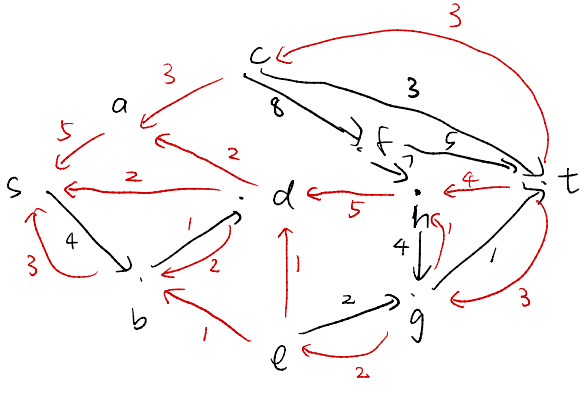
Current flow = 9



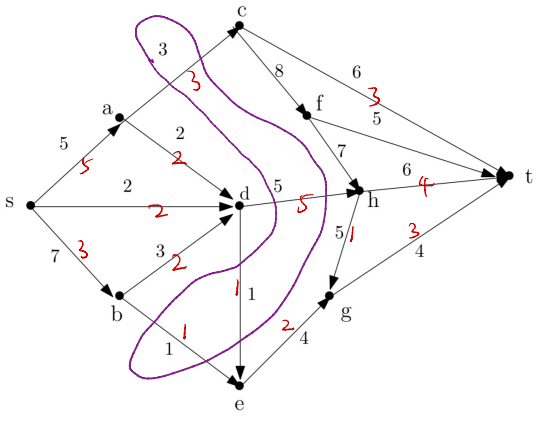
Step 6:



Current flow = 10



(b) final flow = 10



those edges circled is the cut with capacity to the final flow

the cut is : $\{(a,c),(d,h),(d,e),(b,e)\}$