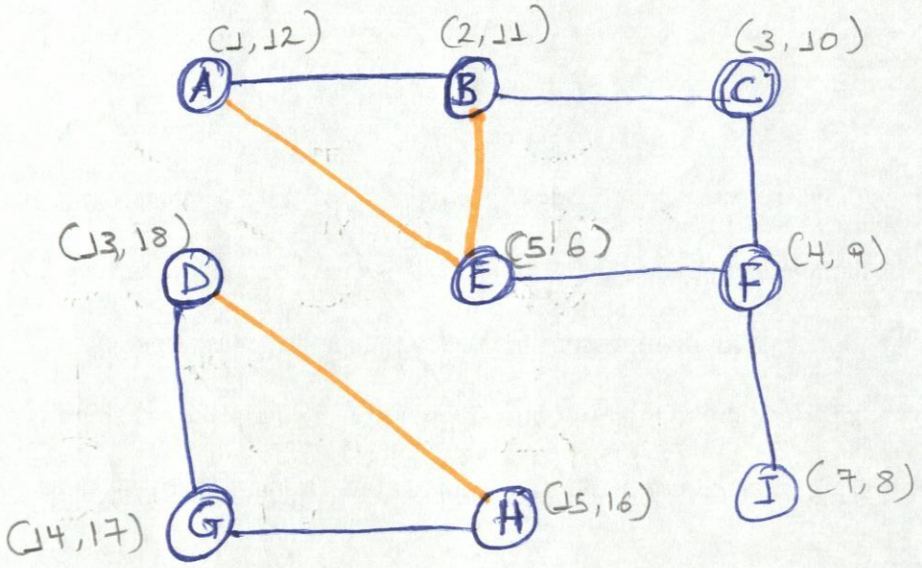
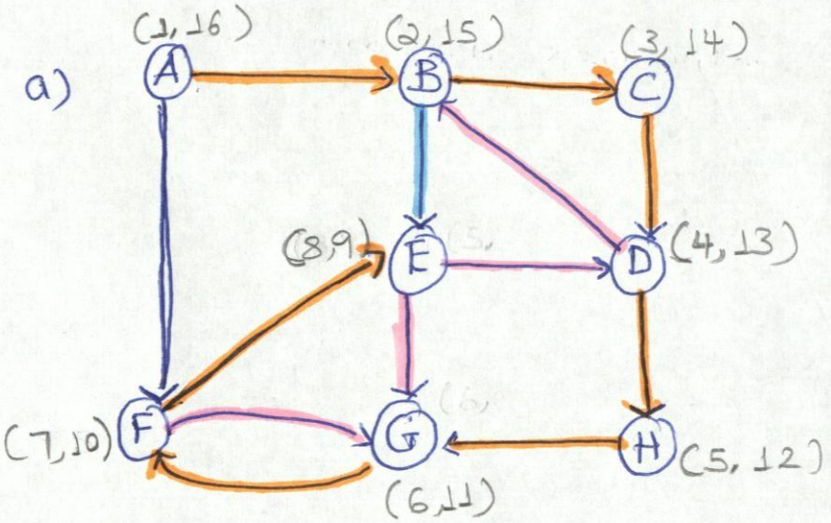
Assignment 4

3.1.



3.2

a)



Tree edges

* A-B
* B-C
* C-D
* D-H
* H-G
* G-F
* F-E

Back edges

* D-B
* E-D
* F-G

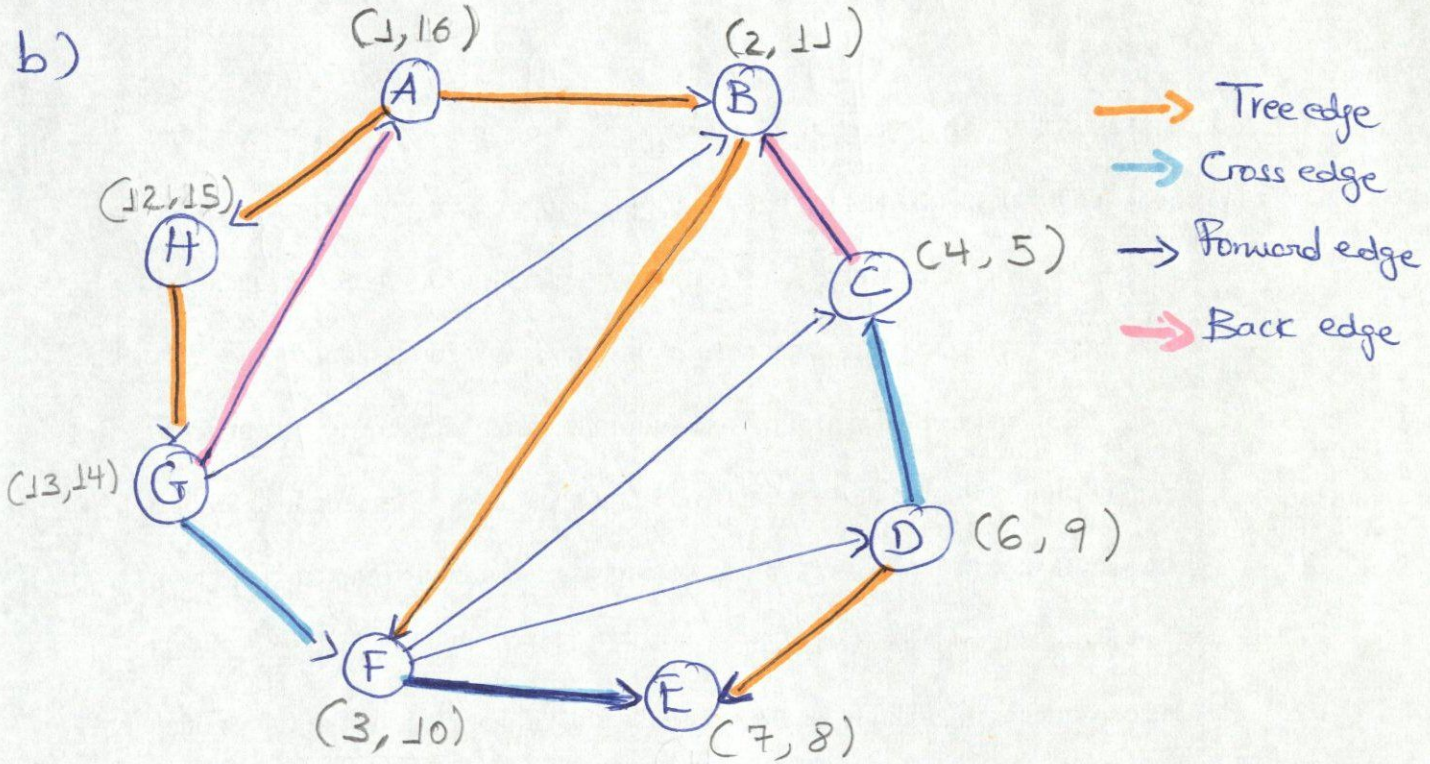
cross edge

* B-E

Forward edge

* A-F

b)



Tree edge

* A - B
* B - F
* D - E
* A - H
* H - G

Back edge

* G - A
* C - B

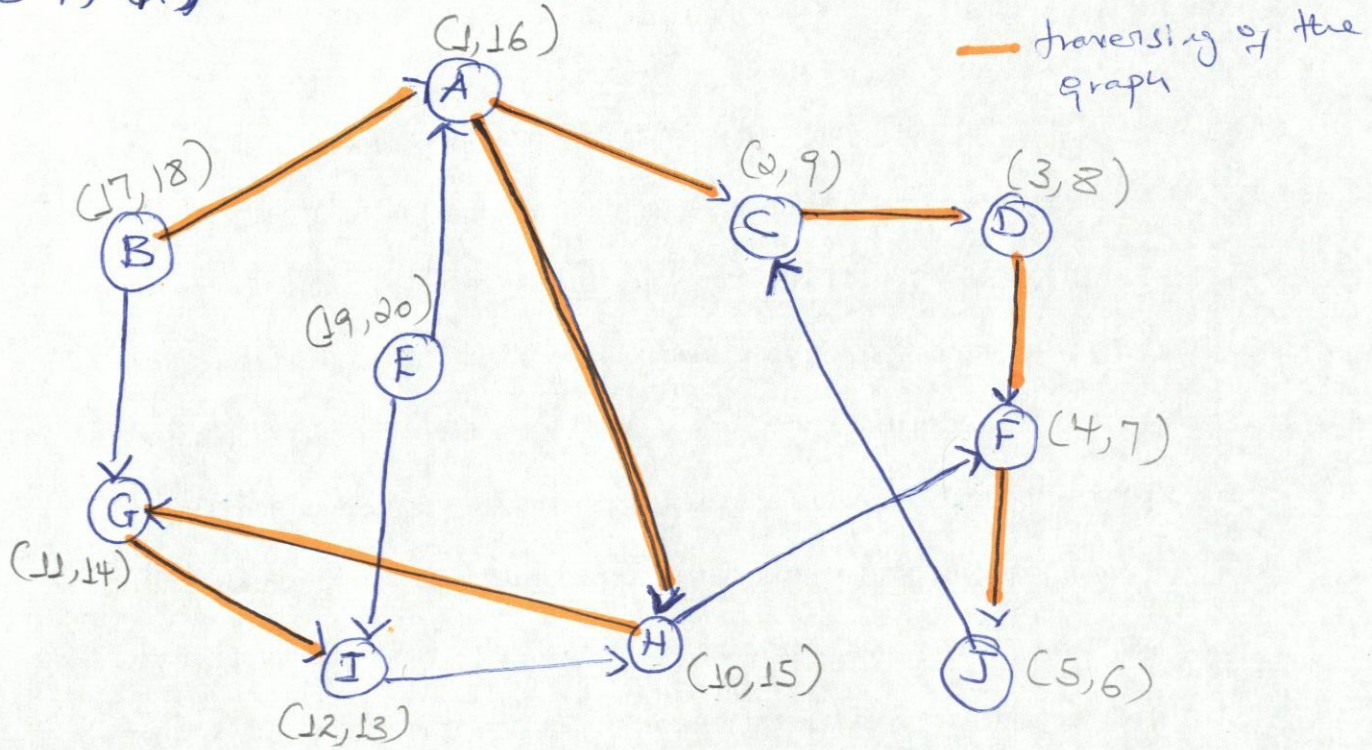
Cross edge

* D - C
* G - F

Forward edge

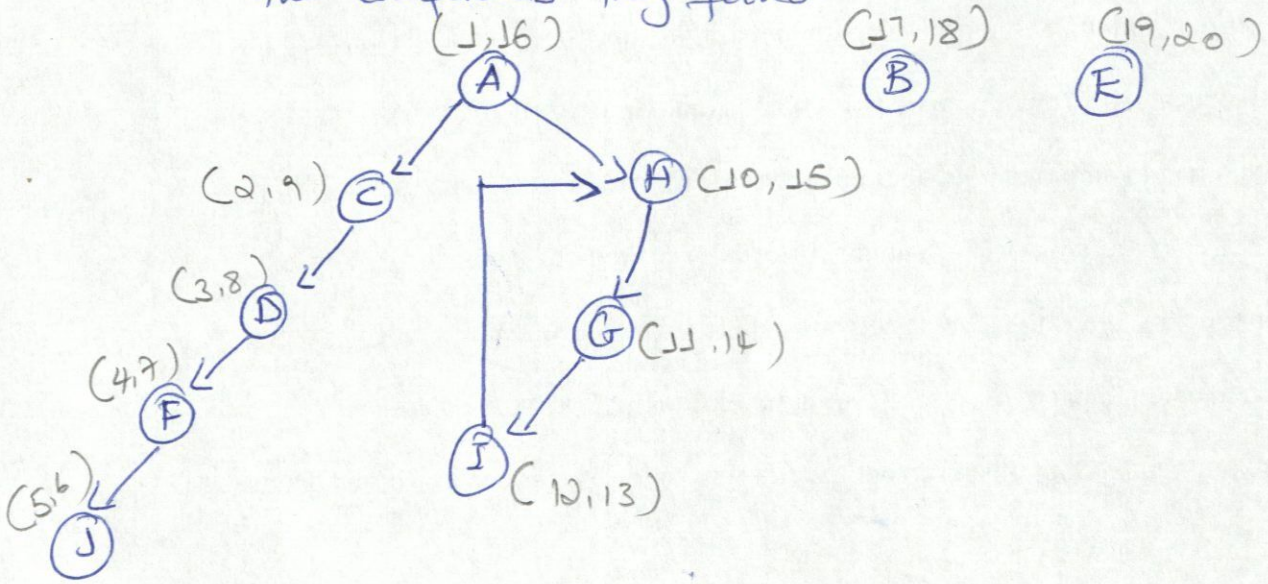
* F - E

3,4) (a))

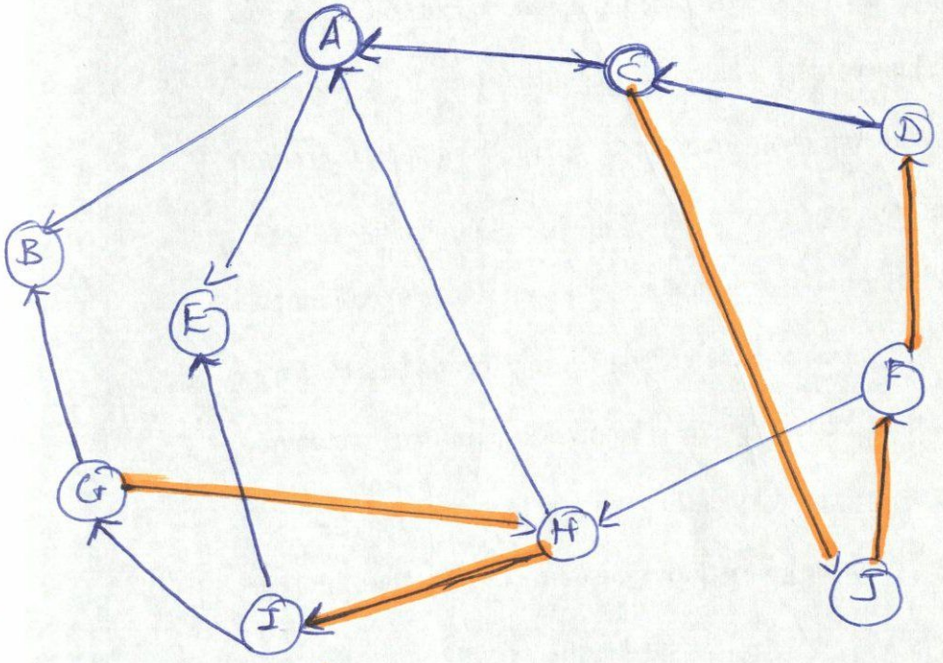


a) In what order are the strongly connected components (SCCs) found?

* Construct the DFS of the given graph as above.
* the first SCC Component is the one whose value is the highest one
* Order the scc in decreasing order, to obtain the order of the components they found.

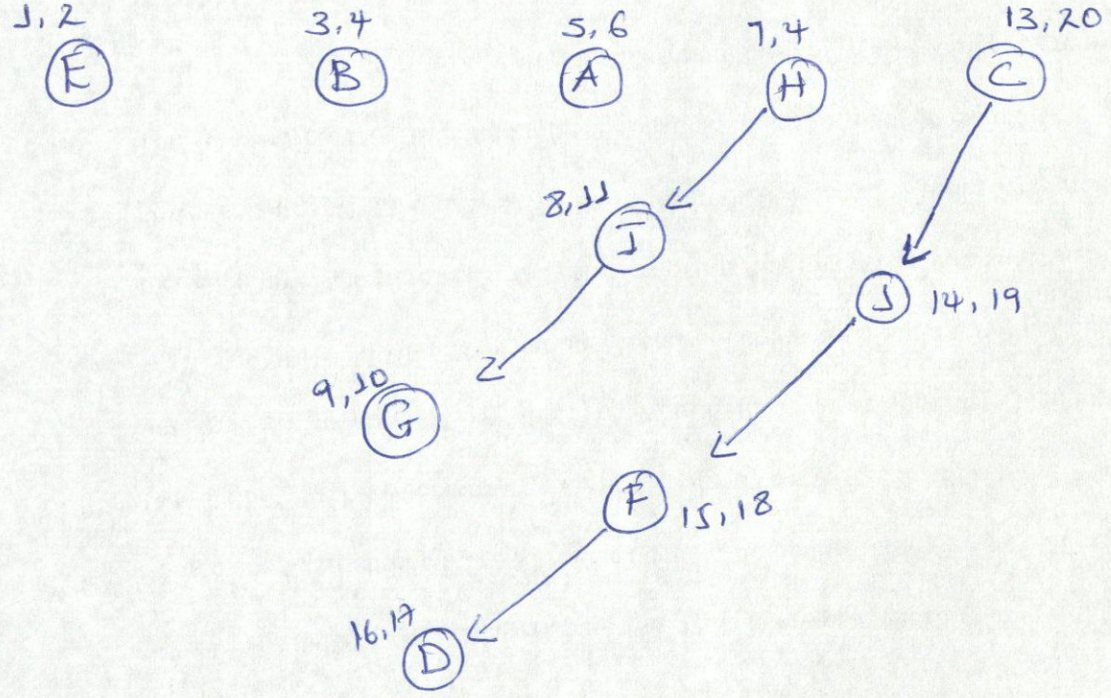


The traversing of the edges is shown above.
The transposed graph is represented by G^R



— Traverse the transported graph G^R.

Now apply DFS on above graph G^R.



To obtain the SCC components, combine both the graph G and G^R. ($G = G^R$)

Therefore the components obtain are as follows.

{C, S, F, D}

{B}

{G, H, I}

{A} and {E}

B. the node with the heighest value is E.

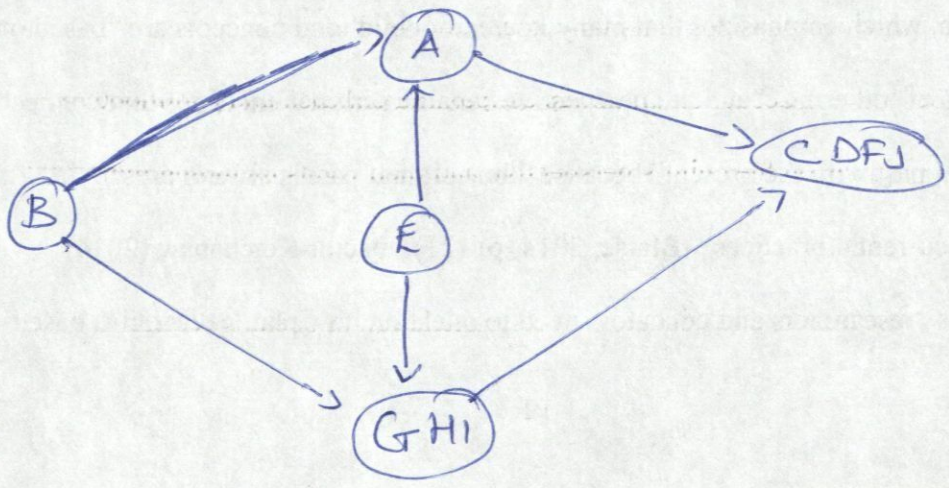
Therefore the component which strongly connected one is the one in which A lies.

Thus, the source strongly Connected Component is {E}.

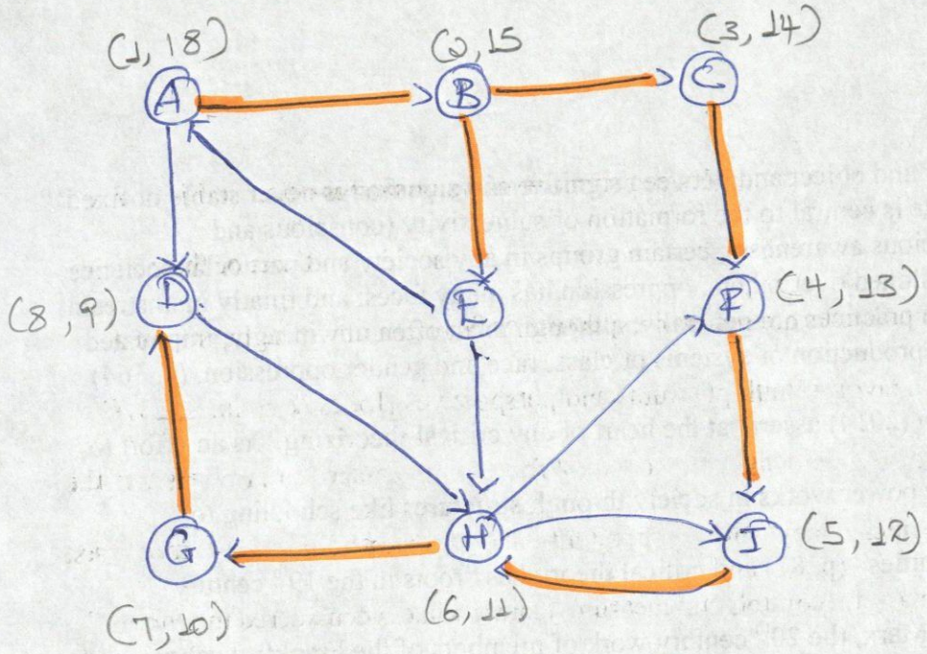
Also; in the above graph DFS graph G^R, the node with the highest value is G. Therefore, the component which is strongly Connected One is the one which A lies.

Thus, the sink strongly Connected Component is {G, H, I}

C. The metagraph is shown below.

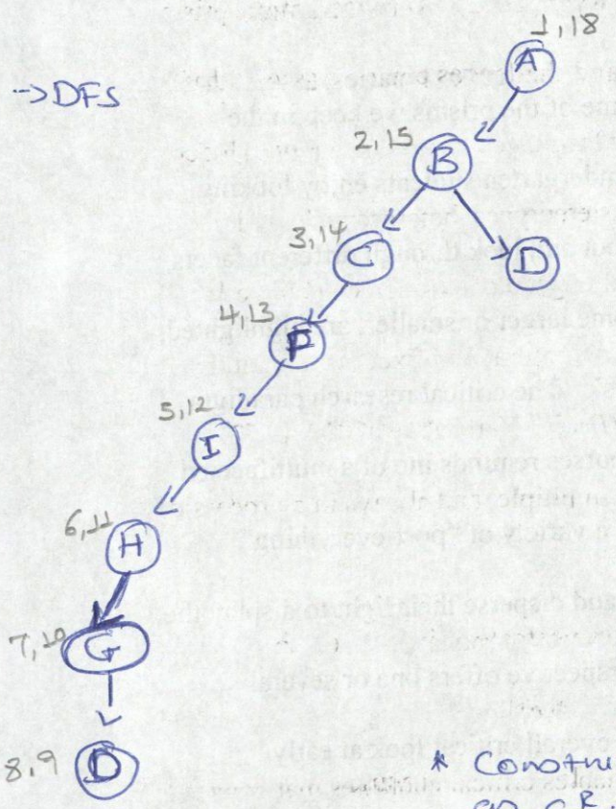


ii)

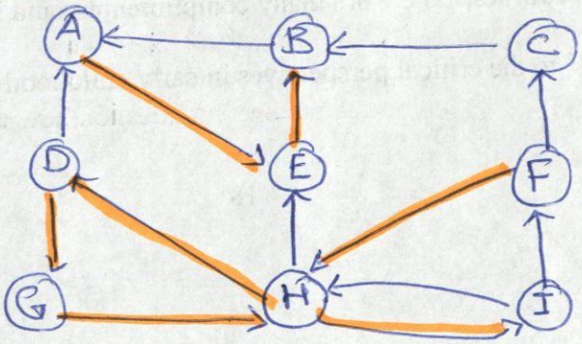


traverse of the graph

->DFS

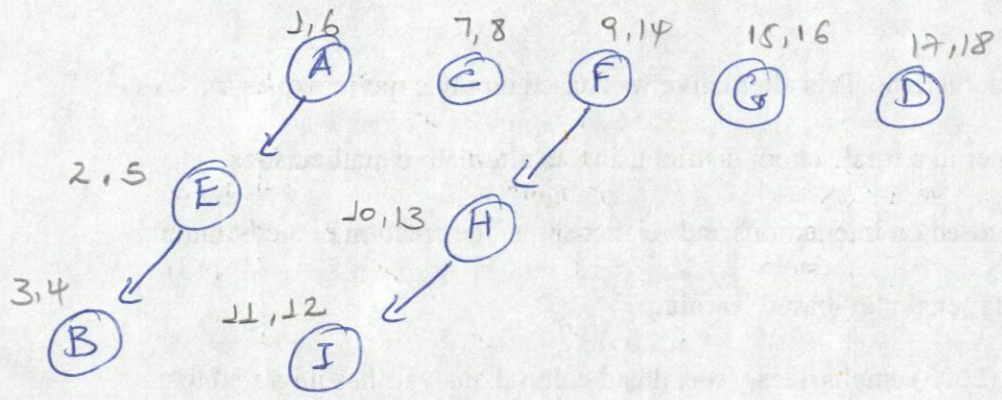


* Construct the transpose of the above graph G as G^R.



traverse of the edges transposed graph G^R

Apply DFS on the above transposed graph.



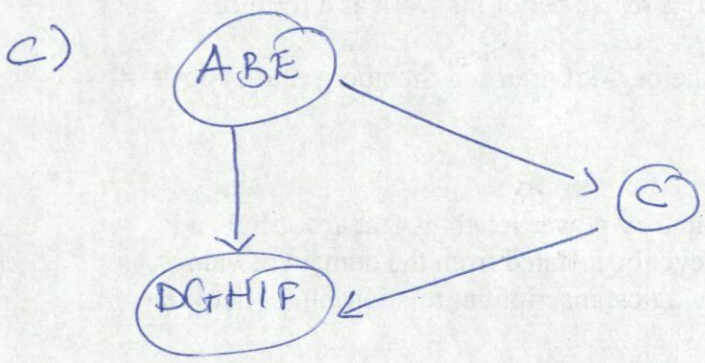
To obtain the scc components, combine both the graph G and G^R ($G_2 G^R$)

Therefore the components obtain are as follows

{D, G, H, I, F} and {A, E, B}

b) In the above DFS graph G, the nodes with the highest value is A. Therefore, the component which is strongly connected one is the one in which A lies. Thus, the source strongly connected Component is {A, E, B}. Also in the graph DFS graph G^R the node with the highest value is 1. Therefore the Component which is strongly connected one is the one in which A lies. Thus, the strongly connected components is {1, H, D, G}.

c)



Ex. 3.8 from the text book

3) a) Let $G = (V, E)$ be our (directed) graph. We will model the set of nodes as triples of numbers (a_0, a_1, a_2) where the following relationships hold: Let $S_0 = 10$ be the size of the corresponding containers.
$S_1 = 7$
$S_2 = 4$

a_i will correspond at the actual contents of the i^{th} container. It must hold $0 \leq a_i \leq S_i$ for $i = 0, 1, 2$ and at any given node $a_0 + a_1 + a_2 = 11$. (the total amount of water we started from).

An edge between two nodes (a_0, a_1, a_2) and (b_0, b_1, b_2) exists if both the following are satisfied. The two nodes differ in exactly two coordinates (and the third one is the same in both). If i, j are the coordinates they differ in, then either $a_i = 0$ or $a_j = 0$ or $a_i = S_i$ or $a_j = S_j$.

b) Given the above description, it is easy to see that a DFS algorithm on that graph should be applied, starting from node $(0, 7, 4)$ with an extra line of code that halts and answers 'Yes' if one of the desired nodes is reached and 'No' if all the connected component of the starting node is exhausted and no desired Vertex is reached.

Ex. 3.16 from the text book

H) Solution

Assume there is a path p in a graph G. Since, each course in p is pre-requisite for the next course in p, the course in p are scheduled in different semester. Therefore, it shows that Numbers of semester are atleast equal to length of p.

Let Q be the longest path in graph G.

The minimum number of semesters will be as big as Q.

Let the length of Q be K.

To design an optimal algorithm, all courses must be scheduled in K semesters. greedy approach can be used to design an optimal algorithm in this case.

Let's prove the optimality of the algorithm. Let's assume that the algorithm operates in rounds. use variable 'j' to represent round. For each round j, if j>1 the algorithm finds all courses that have no pre-requisites, schedules them in semester j, and deletes them from G. The algorithm is terminated when graph G is empty. There will be at least one course without a pre-requisite in a Directed Acyclic Graph (DAG). By deleting a node from DAG, since there can be no cycles the algorithm terminates after K rounds (maximum).

Con't,

To prove that Greedy approach is optimal, it is important to prove that the algorithm terminates in k rounds. To prove above statement do the following.

- prove that in graph G_T, the courses in that path are scheduled with k-semesters. Let d_v be the depth of a course $v \in G_T$ to be the number of courses in the longest path in G_T that terminates in v. We can prove by using Induction.

Let the base case be $d_v = 1$. when $d_v = 1$, the courses with no pre-requisites are found and these courses are scheduled in first semester. for Inductive hypothesis it is enough to prove $d_u \leq 1$ in semester d_u, for all courses u.

prove $d_v = l + 1$ for $l + 1$ with course v. Let G_l be the graph that remains at the end of round l.

Let v be the course with depth $l + 1$ in G_T. It is enough to prove that the depth of v in G_1 is one, since the algorithm is scheduled v in this round $l + 1$. If suppose the depth of v in G_1 is larger than 1, the v will have pre-requisite u in G_1. by the Inductive hypothesis all courses u are scheduled with $d_u \leq l$ for semester d_u. Therefore, d_u must be larger than l which means that the depth of v is $d_v > d_u + 1 > l + 1$.

The above statement is contradiction, because $d_v = l + 1$ which completes the proof.

Running time of the algorithm.

for each course v, maintain a Counter P_v.

See the pre-requisite pairs (u) in G. For round 1, find all courses with $P_v = 0$ by scanning the graph in linear manner.

for each round i for course u do the following.

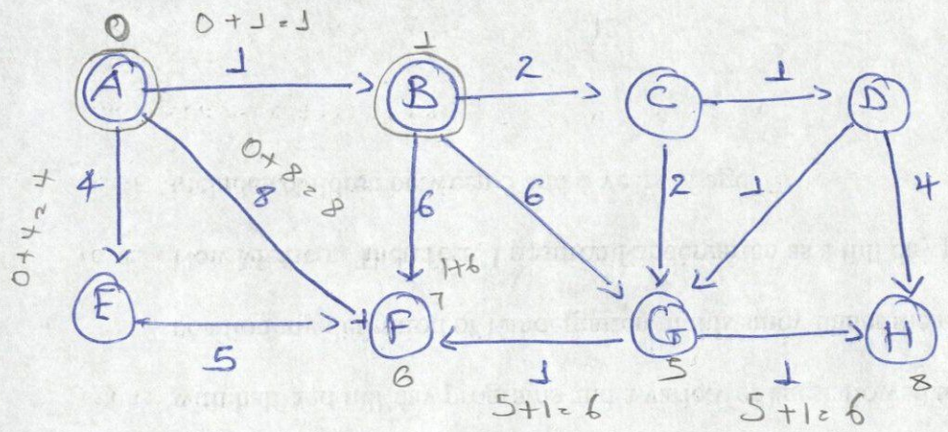
* P_v is decremented by 1 for all courses v, for which u is pre-requisite.
* If P_v reaches to 0, add all nodes which are to be scheduled in the next round.

Since G is provided with adjacency list format, edge(u,v) can be processed only when u is scheduled.

Therefore, the running time of the algorithm is $O(m+n)$.

Ex 4.1 from the text book

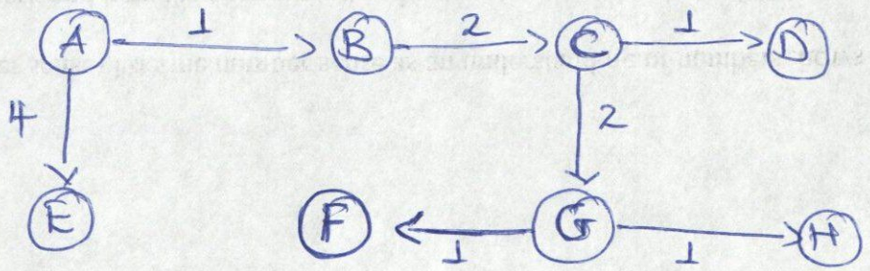
5.



A is the source

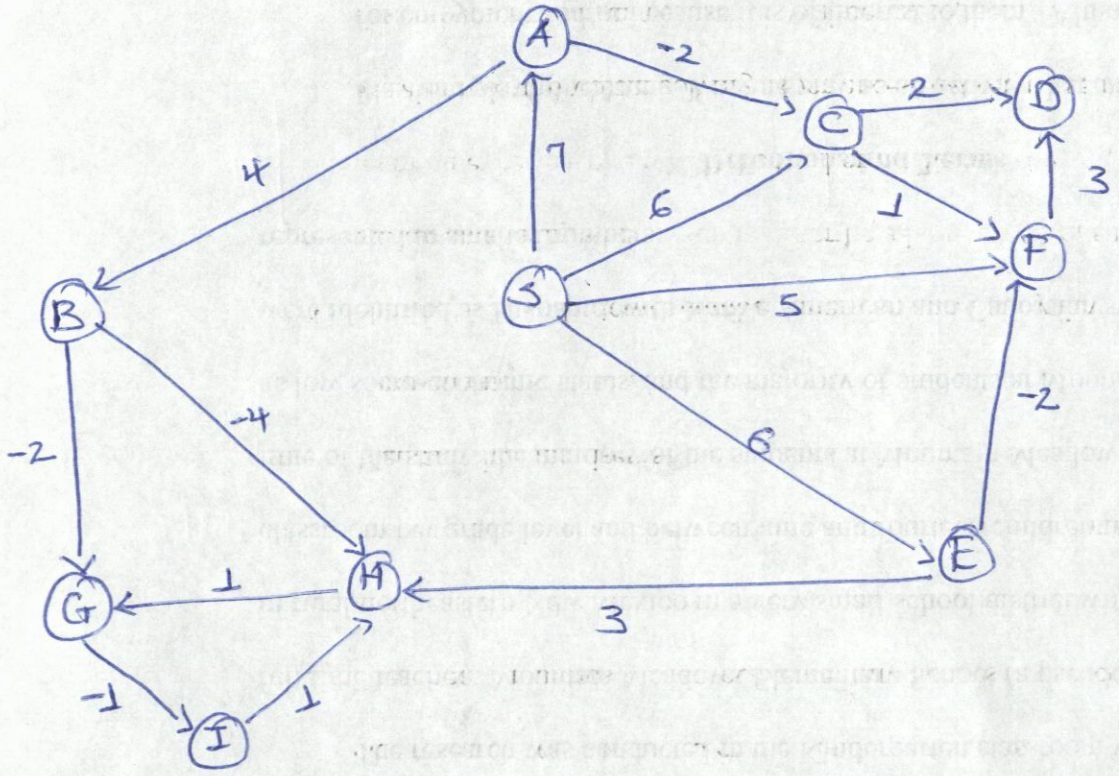
	A	B	C	D	E	F	G	H
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	1	∞	∞	4	8	∞	∞
2	0	1	3	∞	4	7	7	∞
3	0	1	3	4	4	7	5	∞
4	0	1	3	4	4	7	5	8
5	0	1	3	4	4	9	5	8
6	0	1	3	4	4	6	5	6

b) Shortest path tree



Ex 4.2 from the text book

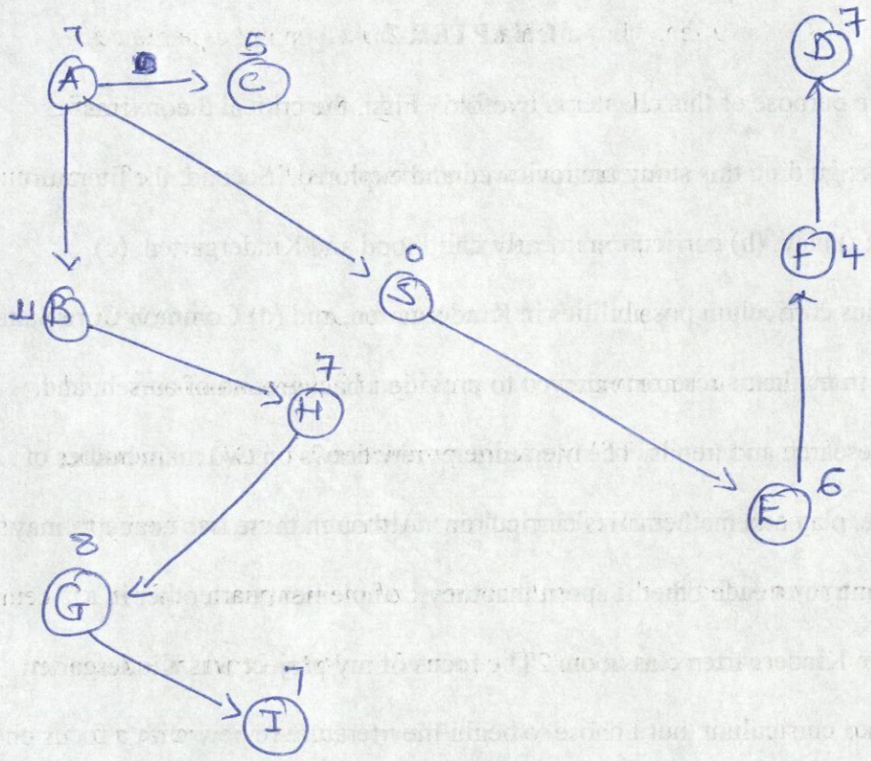
(6) Solution



a)

	S	A	B	C	D	E	F	G	H	I
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	7	∞	6	∞	6	5	∞	∞	∞
2	0	7	11	5	8	6	4	∞	9	∞
3	0	7	11	5	7	6	4	9	7	∞
4	0	7	11	5	7	6	4	8	7	8
5	0	7	11	5	7	6	4	8	7	7

b. The shortest tree



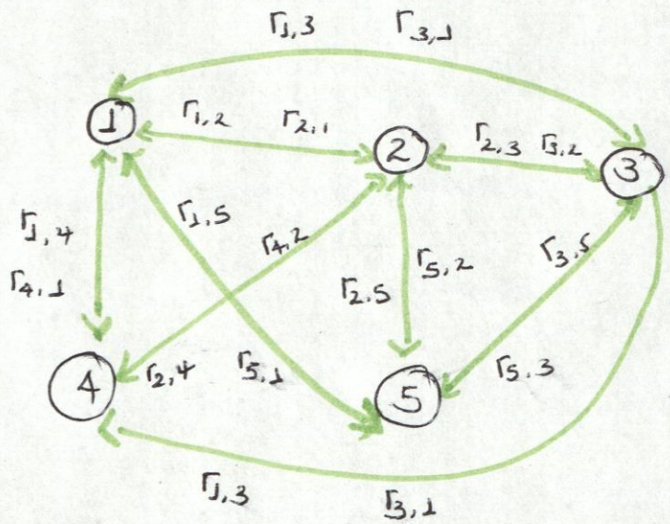
Question 4.21 from the text book.

Solution

a) Let say there are n currencies $C_1 \ldots C_5$. To exchange the currency C_i to C_j, the exchange rate is $r_{i,j}$. The product $r_{i,j} \cdot r_{j,i}$ is always less than 1. to get the most advantage sequence of currency exchanges, Maximize the product of exchange rates $r_{1,2,3} \cdot r_{3,4} \ldots r_{k-1,k}$

Suppose $w_{i,j}$ is the weight of an edge which starts from C_i and connects at C_j. Then $w_{i,j} = -\log r_{i,j}$

To get the maximum exchange rate, minimize the sum of weights. As weights can be negative, use bellman ford Algorithm to find the shortest path to graph, taking any currency as origin.



Graph representing the currency exchange rate

r	1	2	3	4	5
1	0	0.5	0.8	0.6	0.1
2	0.9	0	0.3	0.7	0.3
3	0.7	0.5	0	0.4	0.9
4	0.3	0.6	0.3	0	0.6
5	0.4	0.4	0.7	0.3	0

The table shows exchange rates.

The algorithm to get the most advantageous sequence of currency rates is as follows

Currency-exchange-algo (G_r, w_{ij}, s)

Input : Directed Graph $G = (V, E)$

Edge weights $\{w_e : e \in E\}$ with no negative cycles.

Vertex $s \in V$

output : For all vertices v reachable from s, min_v is set to minimized sum of weights.

For all $v \in V$:

$w(v) = \infty$

prev $(v) = nil$

$w(s) = 0$

repeat $|V| - 1$ times

For all $e \in E$ -> update (e)

b) Iterate the updating procedure once more after $|E||x|$ rounds. If any distance is updated a negative cycle is guaranteed to exist - i.e a cycle with $\sum_{j=1}^{k-1} (-\log r_{i_j, i_{j+1}}) < 0$, which implies $r_{i_1, i_2} \cdot r_{i_3, i_4} \cdot r_{i_5}, \ldots, r_{i_{k-1} \cdot i_2} r_{i_2 \cdot i_1} > 1$ as required.