2018. 1 (a)
What do you mean by Algorithm? What are the basic criteria that an algorithm must satisfy?
-(09) marks

An algorithm is a finite set of instructions which is followed to accomplish a particular task.
An algorithm should follow some criteria:

1. Input : there should be zero or more quantities are externally supplied.
2. Output : one or more quantities are produced from the algorithm
3. Definiteness : each instruction is clear and unambiguous
4. Finiteness : the algorithm should terminate after a finite number of steps.
5. Effectiveness : every instruction must be very basic so that it can be carried out by a person using only pen and pencil.

b) Consider the code segment of addition of two matrices mentioned in the following figure. (10)

```
int **matrix_sum(int m1[][3],int m2[][3]){
 int i, j, **m3;
 m3 = malloc(sizeof(int*)*3);
 for(i=0;i<3;i++)
 {
   for(j=0; j<3; j++)
   m3[i][j]=m1[i][j]+m2[i][j];
 } return m3;}
```

Let int takes 2 bytes in memory. Now, show the step counts of the above program with respect to space and time, where the unit of space and time, where the unit of time is m sec.

2018. 1 (b)
couldn't solve :(

c) Write down the best Big-oh (o) characterization for each of the running time estimates of (16) different algorithms: (i) log(n!), (ii) 6.2^n+n^2 (iii) $1000n^2+100n-6$ (iv) $6n^3/(\log n+1)$.

2018. 1 (c)

(2018) 1(c) i) $\log(n!)$

$\log(n!) = \log(n.(n-1).(n-2)\ldots 3.2.1)$

$= \log n + \log(n-1) + \ldots \log(2) + \log(1)$

Which is less than

$\log n + \log n + \log n + \ldots + \log n = n * \log n$

So $O(\log(n!)) \subseteq O(n \log n)$

$f(n) \leq c \cdot g(n)$

$\log n + \log(n-1) + \ldots + \log(2) + \log(1) \leq \log n + \log n + \log n + \ldots + \log n$

$\log n + \log(n-1) + \ldots + \log(2) + \log(1) \leq n \log n$

$\therefore f(n) = O(n \log n)$

(2018) 1© ii) ~~$6.2^n + n^2$~~ $f(n) = 6.2^n + n^2$

$f(n) \leq c.g(n)$

$6.2^n + n^2 \leq 6.2^n + 2^n$

$6.2^n + n^2 \leq 7.2^n$

$\therefore f(n) = O(2^n)$

1©(iii) $f(n) = 1000n^2 + 100n - 6$

$f(n) \leq c.g(n)$

$1000n^2 + 100n - 6 \leq 1000n^2 + 100n^2 - 6n^2$

$1000n^2 + 100n - 6 \leq 1094.n^2$

$\therefore f(n) = O(n^2)$

1©(iv) $f(n) = \frac{6n^3}{\log n + 1}$

$f(n) = c.g(n)$

To be continued

Hotasha

2. a) The complexity of a merge sort algorithm can be described as follows: (15)

$$\begin{cases} a & ,n=1 \\ 2T\left(\frac{n}{2}\right)+cn, & n>1 \end{cases}$$

Here a, c are two constants. Derive an appropriate best case notation for the above recurrence relation.

2018. 2(a)

(2018) 2(a) $T(n) = \begin{cases} a & ; n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & ; n > 1 \end{cases}$

$T(n) = 2T\left(\frac{n}{2}\right) + n$ ——— (1)

$= 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$

$= 2^2 T\left(\frac{n}{2^2}\right) + n + n$

$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2n$ ——— (2)

$T(n) = 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + 2n$

$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 3n$ ——— (3)

. . . . . .

. . . . . .

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$

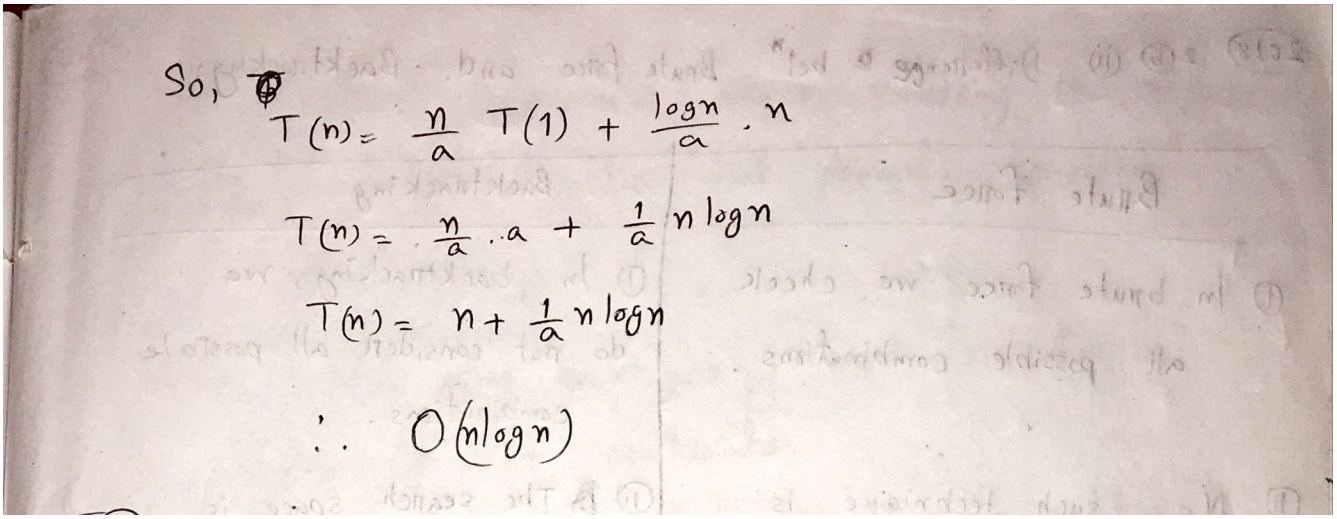
Assume, $T\left(\frac{n}{2^k}\right) = T(1)$

$\frac{n}{2^k} = a$

$n = 2^k \cdot a$

$\log n = ka$

$k = \frac{\log n}{a}$



2. (b) Differentiate between the followings: (i)Brach and bound vs Backtracking (ii) Brute-force vs Backtracking. -(08) marks

2018. 2 (b)
(i)

Branch and Bound	Backtracking
1. It is used to solve optimization problem.	1. It is used to find all possible solutions available to a problem.
2. It may traverse the tree in any manner, DFS or BFS.	2. It traverses the state space tree by DFS manner.
3. It realizes that it already has a better optimal solution that the pre-solution leads to, so it abandons that pre-solution.	3. It realizes that it has made a bad choice & undoes the last choice by backing up.
4. It completely searches the state space tree to get optimal solution.	4. It searches the state space tree until it has found a solution.
5. It involves a bounding function.	5. It involves feasibility function.

(ii)

Brute Force	Backtracking
1. In brute force we check all possible combinations.	1. In backtracking we do not consider all possible combinations.
2. No such technique is applied.	2. The search space is restricted after a failure.
3. Finds best and feasible solution by performing exhaustive search i.e. checks whether every candidate satisfies the constraints.	3. Checks if solution is correct or not at every step. If it is correct, continues generating subsequent solutions. If incorrect backtracks to previous step and checks for other solution.
4. Can be very slow.	4. When applicable, backtracking is much faster than brute force.

2018. 2 (c)

2. (c) What is constrain? How the constraints are useful in Backtracking method? Show the constraints of N-Queen Problem and applying these constraints write a recursive backtracking algorithm to solve N- Queen problem. -(12) marks

Constraint means something that imposes a limit or restriction that prevents something from occuring.
Every solution using backtracking is must satisfy a complex set of constraints.
If a constraint is not satisfied during the process of building up a solution, we reject all possible ways of extending the current partial assignment. The algorithm searches a tree of partial assignments.

Constraints of N-Queens problem:

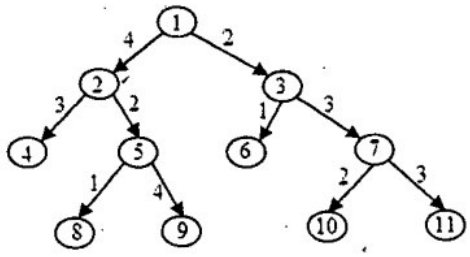
– Can’t put two Queens in same column
$Qi \neq Qj$ for all $i \neq j$
– Diagonal constraints
$|Qi-Qj| \neq i-j$

i.e. the difference in the values assigned to Qi and Qj can’t be equal to the difference between i and j.

Algorithm for N-queen problem:

1: Let n be the total number of queens, let us denote the queen number by k. k=1,2,..,n.
2: We start a loop for checking if the k'th queen can be placed in the respective column of the k'th row.
3: For checking that whether the queen can be placed or not, we check if the previous queens are not in diagonal or in same row with it.
4: If the queen cannot be placed backtracking is done to the previous queens until a feasible solution is not found.
5: Repeat the steps 2-4 until all the queens are placed.
Step 7: The column numbers of the queens are stored in an array and printed as a n-tuple solution
Step 8: Stop

3. a) Consider the graph mentioned in following figure (13)
Now split the vertex of the graph using greedy method.

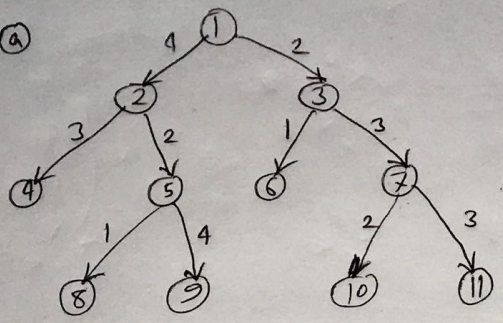


2018. 3 (a)

solution : If u has a parent v such that
d(u) + w(v, u) > limit, then the node u gets split and d(u) is set to 0.
d(u) = max { $d(v) + W(u, v)$}
$v \in C(u)$
where C(u) is the set of all children of u.

reference: https://www.youtube.com/watch?v=rvncntb3iNU

2018 3 (a)



Lets assume, limit = 5.

If u has a ~~per~~ parent such that $d(u) + w(v,u) > limit$,

then the node u gets split ~~int~~ and d(u) is set to 0.

$d(u) = \max\{d(v) + w(u,v)\}$

$v \in C(u)$ where $C(u)$ is the set of all children of u.

Here,

Limit = 5

~~d(4)~~ = ~~d(2) = max (3,2) + w(2,4)~~

$d(2) = \max(3,2) + w(1,2)$

$= 3 + 4$

$= 7$ which is > limit, so node 2 is split and d(2) = 0

$d(5) = \max(1,4) + w(2,5)$

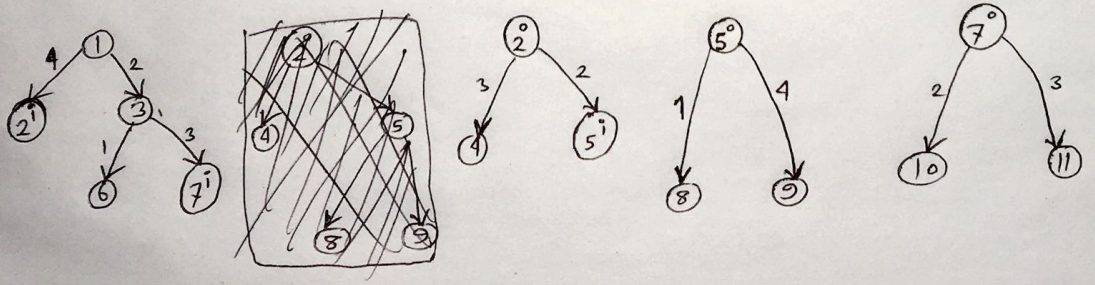
$= 4 + 2$

$= 6$ which is > limit so node 5 is split and d(5) = 0

$d(7) = \max(2,3) + w(3,7)$

$= 3 + 3$

$= 6$ which is > limit so node 7 is split and d(7) = 0



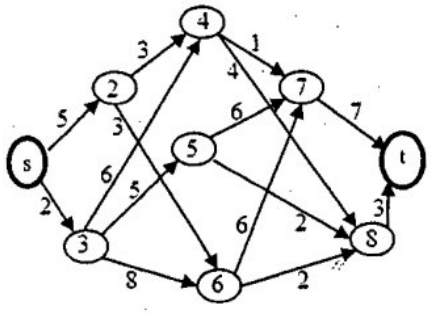
b) Write down the control abstraction of greedy method. (07)

2018. 3 (b)

Control Abstraction of Greedy Method

```
Algorithm Greedy(a,n)
// a[1:n] contains n inputs
{
        solution := 0;
        for i :=1 to n do
        {
                x := select(a);
                if feasible(solution, x) then
                solution := Union(solution,x);
        }
        return solution;
}
```

c) What is principle of optimality in Dynamic programming? Consider the graph mentioned in the following figure:
Using forward approach find a minimum-cost path from s to t in the above multistage graph. (15)



2018. 3 (c)

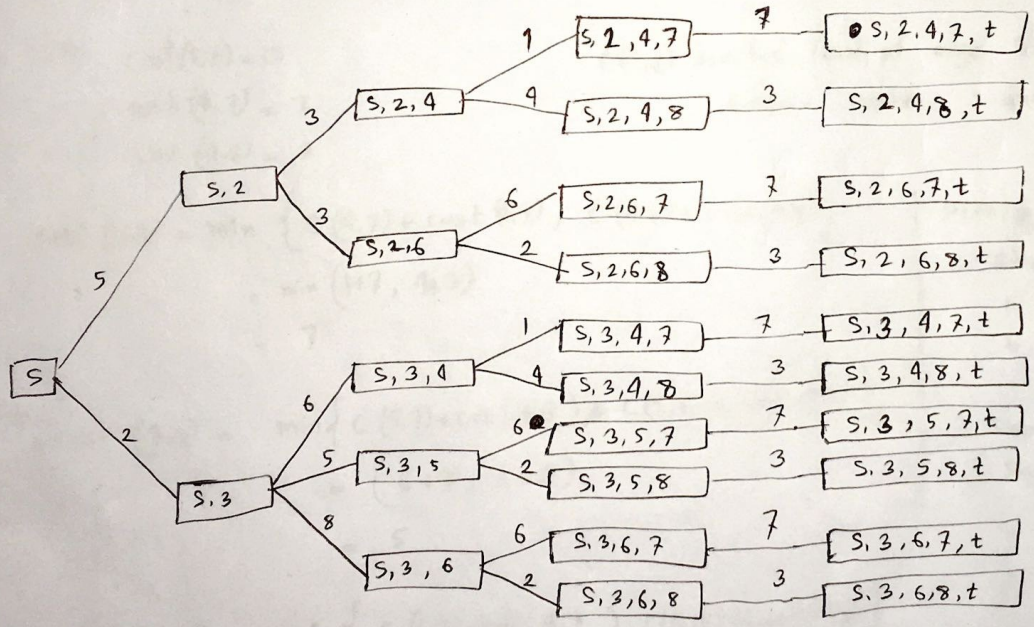
solution : A problem is said to satisfy the Principle of optimality if the subsolutions of an optimal solution of the problem are themesleves optimal solutions for their subproblems.
The principle of optimality is the basic principle of dynamic programming.

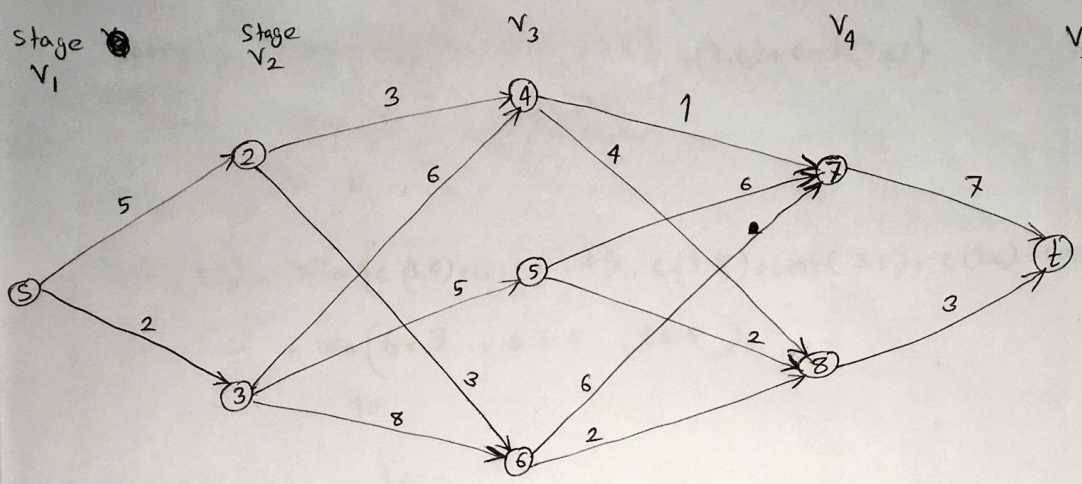
The shortest path problem satisfies the Principle of optimality.

(2018) 3©

	S	2	3	4	5	6	7	8	t
S	-	5	2	-	-	-	-	-	-
2	-	-	-	3	-	3	-	-	-
3	-	-	-	6	5	8	-	-	-
4	-	-	-	-	-	-	1	4	-
5	-	-	-	-	-	-	6	2	-
6	-	-	-	-	-	-	6	2	-
7	-	-	-	-	-	-	-	-	7
8	-	-	-	-	-	-	-	-	3
t	-	-	-	-	-	-	-	-	-

M





cos(i,j) denotes shortest path of j^{th} vertex of i^{th} stage, from the destination

cost(5,t) denotes shortest path of vertex t of stage 5, from the destination t, which is 0

Here, cost(5,t) = 0
cost(4,7) = 7
cos(4,8) = 3

c(i,j) denotes cost of edge ~~i and~~ j between vertex i and j

$$\text{cost}(3,4) = \min\{c(4,7) + \text{cost}(4,7),\ c(4,8) + \text{cost}(4,8)\}$$
$$= \min(1+7,\ 4+3)$$
$$= 7$$

$$\text{cost}(3,5) = \min\{c(5,7) + \text{cos}(4,7),\ c(5,8) + \text{cost}(4,8)\}$$
$$= (6+7,\ 2+3)$$
$$= 5$$

$$\text{cost}(3,6) = \min\{c(6,7) + \text{cos}(4,7),\ c(6,8) + \text{cost}(4,8)\}$$
$$= \min(6+7,\ 2+3)$$
$$= 5$$

Here,
stage 1 = V_1
" 2 = V_2
" 3 = V_3
~~And so on~~
stage 4 = V_4
stage 5 = 5

$$cost(2,2) = \min\{ c(2,4) + cost(3,4), c(2,6) + cost(3,6) \}$$

$$= \min(3+7, 3+5)$$

$$= 8$$

$$cost(2,3) = \min\{ c(3,4) + cost(3,4), c(3,5) + cost(3,5), c(3,6) + cost(3,6) \}$$

$$= \min(6+7, 5+5, 8+5)$$

$$= 10$$

~~$cost(1,s) = \min\{c(s,z)$~~

$$cost(1,s) = \min\{ c(s,2) + cost(2,2), c(s,3) + cos(2,3)$$

$$= \min(5+8, 2+10)$$

$$\boxed{cos(1,s) = 12}$$

$cost(1,s)$ is the shortest path of vertex s of stage 1, from the destination t, which we are looking for

Shortest Path : cost 12

$s \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow t$

4. a) What do you mean by approximation algorithm? Why do we need approximation algorithm? (09)

2018. 4 (a)

Approximation algorithms are efficient algorithms that find approximate solutions to optimization problems (in particular NP-hard problems)
Approximation algorithms are often associated with NP-hard problems; since it is unlikely that there can ever be efficient polynomial-time exact algorithms solving NP-hard problems, one settles for polynomial-time sub-optimal solutions.

- An approximation algorithm guarantees to run in polynomial time though it does not guarantee the most effective solution.
- An approximation algorithm guarantees to seek out high accuracy and top quality solution
- Approximation algorithms are used to get an answer near the (optimal) solution of an optimization problem in polynomial time

Approximation algorithms are increasingly being used for problems where exact polynomial-time algorithms are known but are too expensive due to the input size.

b) Define NP-hard and NP-complete problems? Prove that clique problem is NP-complete. (06)

2018. 4 (b)

Np hard

Is a class of problems that are, at least as hard as the hardest problems in NP. More precisely, a problem H is NP-hard when every problem L in NP can be reduced in polynomial time.

NP complete

a problem is NP-complete when:

> it is a problem for which the correctness of each solution can be verified quickly and a brute-force search algorithm can actually find a solution by trying all possible solutions.

The *complexilty class* of *decision problems* for which answers can be checked for correctness, given a *certificate*, by an algorithm whose run time is *polynomial* in the size of the input (that is, it is *NP*) and no other NP problem is more than a polynomial factor harder. Informally, a problem is NP-complete if answers can be verified quickly, and a quick algorithm to solve this problem can be used to solve all other NP problems quickly.

c) Why do we need process mapping in parallel algorithm? Map processes using intermediate data decomposition for the following operation: (12)

$$\left[\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \times \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} + \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}\right]^T = \begin{pmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{pmatrix}$$

2018. 4 (c)
couldn’t solve :(

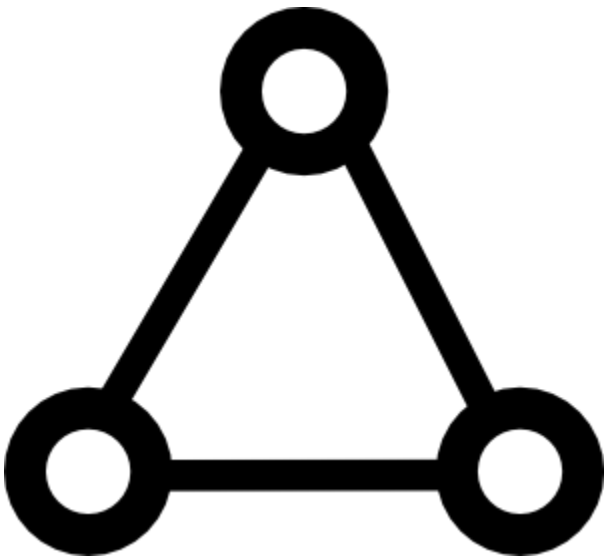
d) “Is the minimum spanning tree generated using Krushkal’s and Prims unique? Explain your answer with example.” (08)

2018. 4 (d)

Prim's and Kruskal's algorithms will always return the same Minimum Spanning tree. A graph where every edge weight is unique (there are no two edges with the same weight) has a unique MST.

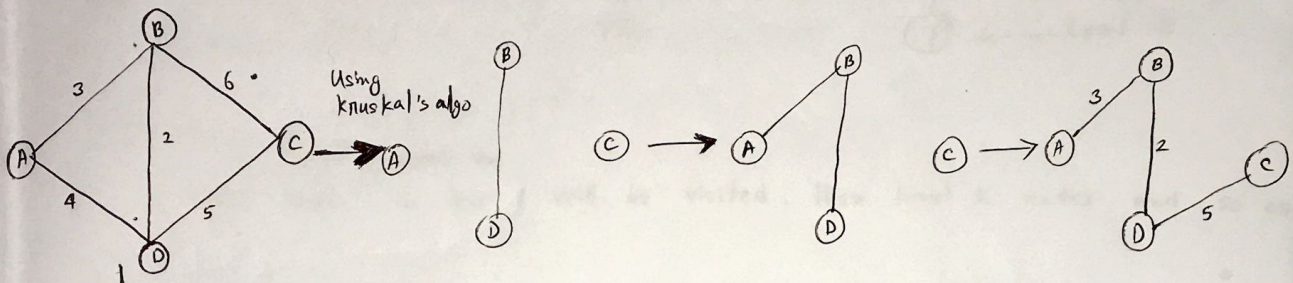
If the MST is unique, all algorithms will produce it.

If the MST is not unique, the outputs might differ due to different node processing orders (even two distinct implementations of the same algorithm can), but the total weights will be identical.



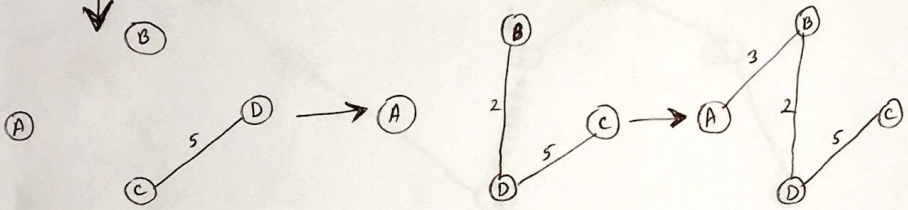
In this graph, we have 3 nodes and 3 edges, each has the same weight. Obviously any 2 edges will form a MST for this graph. However, which two edges are chosen will depend on not only the algorithm, but the implementation of the algorithm.

(2018) 4(d) The minimum spanning tree generated using Kruskal's and prim's algorithm ~~are~~ is same, ~~however~~ however if there are same weight edges in the graph, the MST can be different.

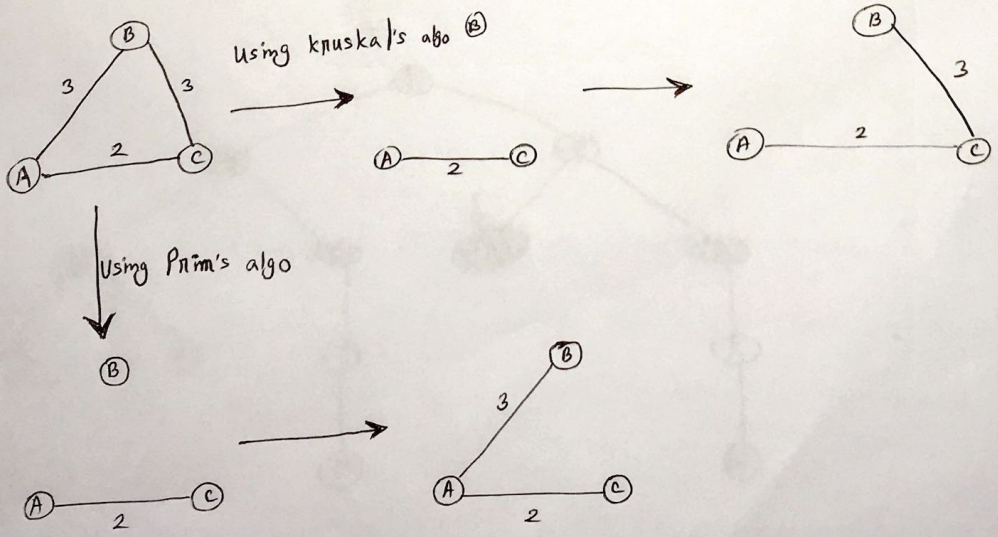


Using Prim's algo

Using Kruskal



Lets take anothe example,



5. a) Graphs can be represented using adjacency Lists or adjacency matrices. What are the memory requirements of each representation? Discuss some advantages and disadvantages for each representation. (10)

2018. 5 (a)

Adjacency List: Representing a graph with adjacency lists combines adjacency matrices with edge lists. For each vertex i, store an array of the vertices adjacent to it. We typically have an array of $|V|$ adjacency lists, one adjacency list per vertex.

This representation can also be used to represent a weighted graph.

Adjacency Matrix: Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.

If 2D array be adj[][], adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

Adjacency Matrix

- Uses O(n^2) memory
- It is fast to lookup and check for presence or absence of a specific edge between any two nodes O(1)
- It is slow to iterate over all edges
- It is slow to add/delete a node; a complex operation O(n^2)
- It is fast to add a new edge O(1)

Adjacency List

- Memory usage depends more on the number of edges (and less on the number of nodes),
  which might save a lot of memory if the adjacency matrix is sparse
- Finding the presence or absence of specific edge between any two nodes
  is slightly slower than with the matrix O(k); where k is the number of neighbors nodes
- It is fast to iterate over all edges because you can access any node neighbors directly
- It is fast to add/delete a node; easier than the matrix representation
- It fast to add a new edge O(1)

Disadvantages of Adjacency Matrix Representation

1. The size of adjacency matrix is V^2. Suppose there is a graph with 1000 vertices and 1 edge. We are using an array of size 1000^2 for storing one edge which is a waste of memory.
2. Traversing the graph using algorithms like DFS / BFS requires $O(V^2)$ time in case of adjacency matrix whereas we can traverse the graph in O(V+E) time using adjacency list.

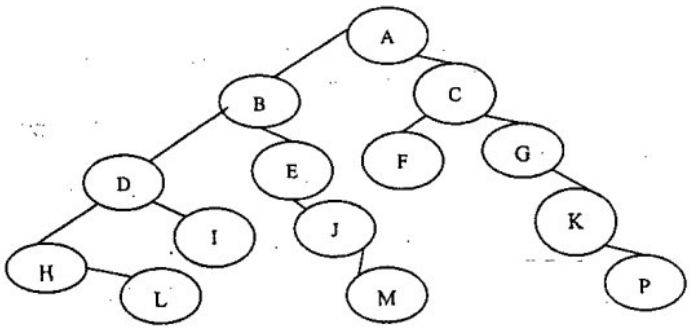
When to use Adjacency Matrix?

1. If the graph contains edges in order of V^2, then it is better to use adjacency matrix as compared to adjacency list. This is because the size of both adjacency list and adjacency matrix will be comparable so using adjacecny matrix doesn't neccessary waste a lot of memeory.
2. If we want to perform operations like add/delete or check that the vertices are adjancent or not very frequently, then it is recommended to use adjacency matrix since we can perform these operations in constant time.

Source : https://stackoverflow.com/questions/2218322/what-is-better-adjacency-lists-or-adjacency-matrices-for-graph-problems-in-c

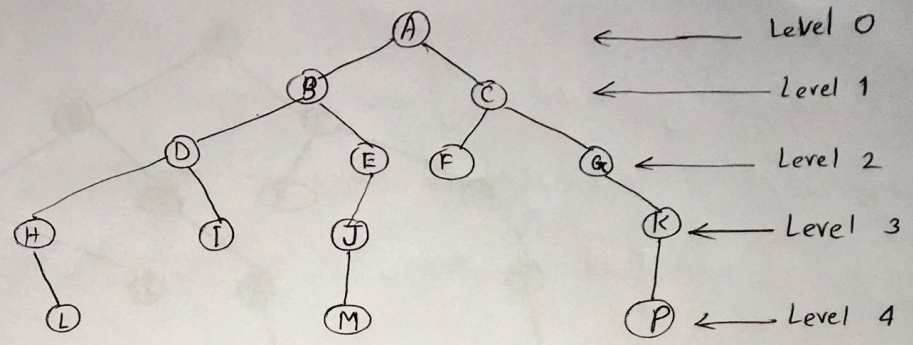
b) In the following graphs, starting from the node A at the top, which algorithm will visit the least number of nodes before visiting the node F. Breadth First Search or Depth First Search? Briefly explain the process to visit F node from node A.

(13)



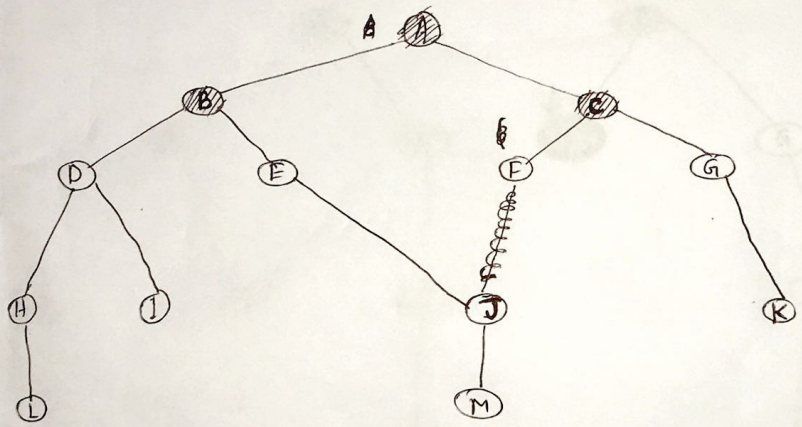
2018. 5 (b)

2018 5 (b)

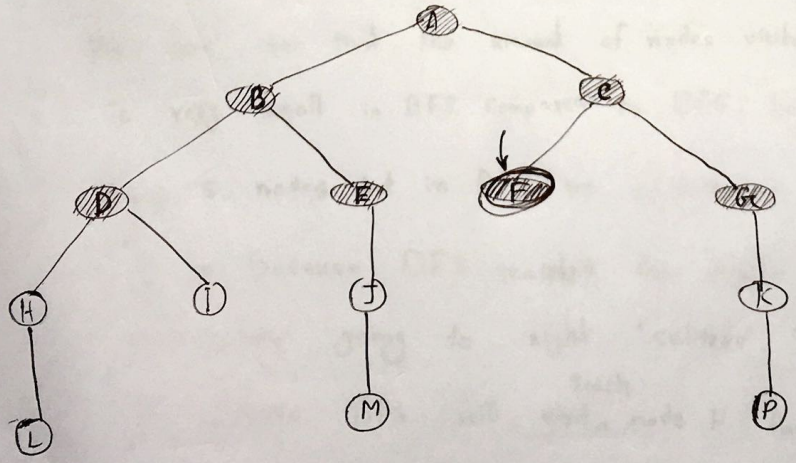


For BFS, ~~first level no~~

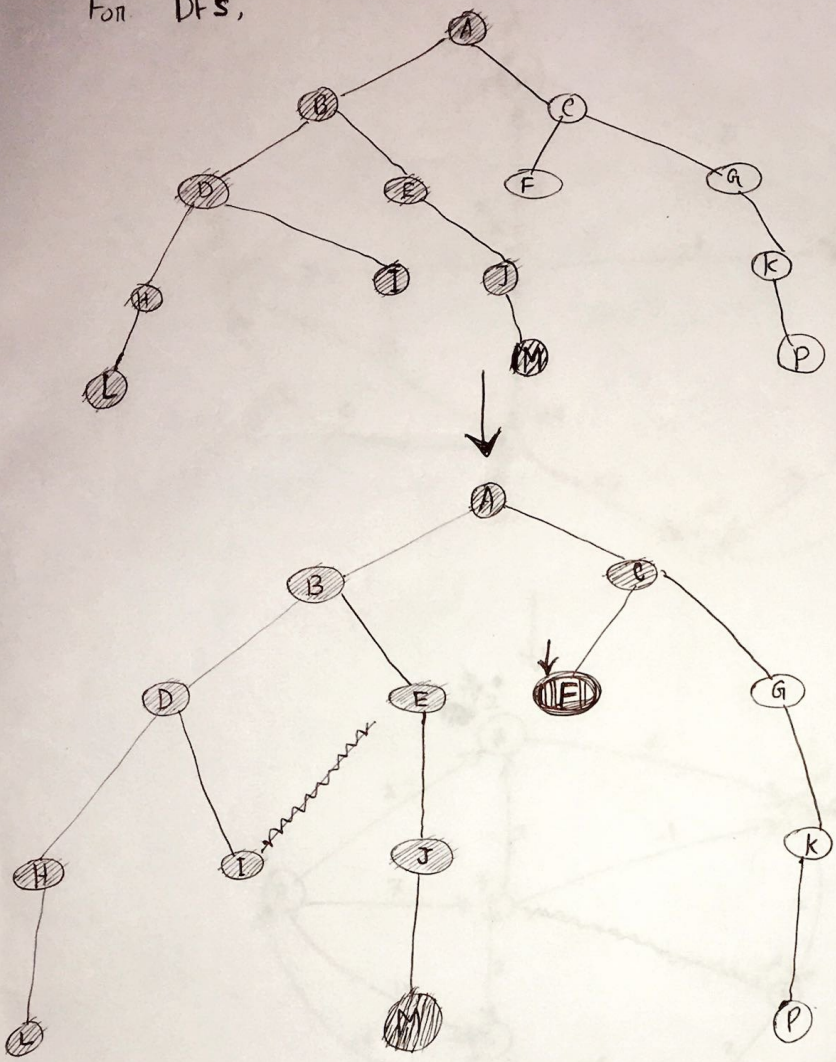
at first nodes in level 1 will be visited, then level 2 nodes and so on.



[after 1st full iteration]



For DFS,



We can see that the amount of nodes visited to reach node F, is very small in BFS compared to DFS. In BFS we found node F after 5 nodes, but in DFS we needed 10 nodes.

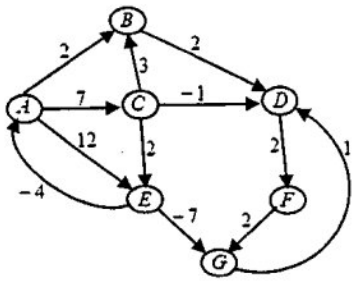
This is because DFS searched the whole left subtree of node A before going to right subtree. So in worst case scenario BFS will reach node F faster.

c) In scheduling independent tasks problem let the number of processor, $m = 3$ number of tasks $n = 7$, where $(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (5, 2, 2, 4, 3, 5, 3)$. Schedule the tasks by Longest processing Time (LPT) rule and then find the time difference between LPT scheduling and optimal scheduling. (12)

2018. 5 (c)

couldn't solve :(

6. a) Consider the following directed, weighted graph. (20)



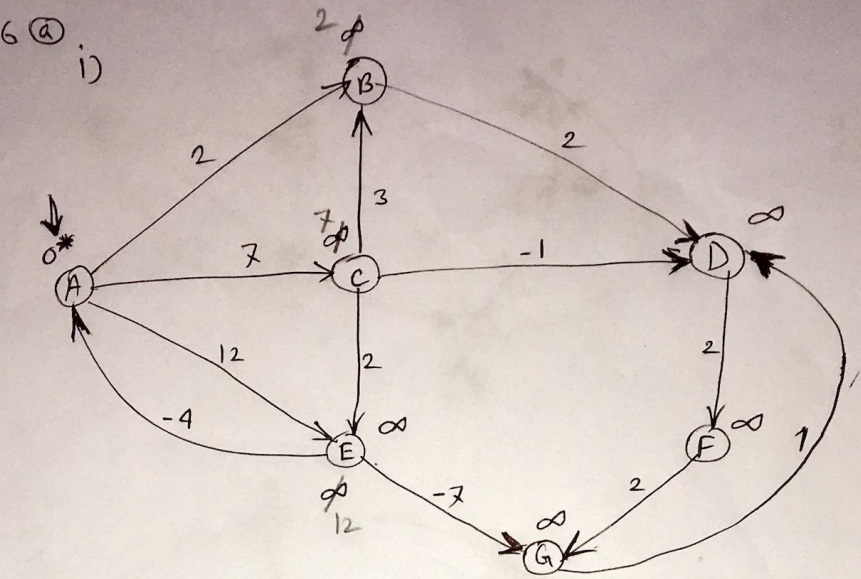
i) Even though the graph has negative weight edges, step through Dijkstra's algorithm to calculate shortest paths from A to every other vertex

ii) Dijkstra's algorithm found the wrong path to some of the vertices. For just the vertices where the wrong path was computed, indicate both the path that was computed and the correct path.

iii) What single edge could be removed from the graph such that Dijkstra's algorithm would happen to compute correct answers for all vertices in the remaining graph.

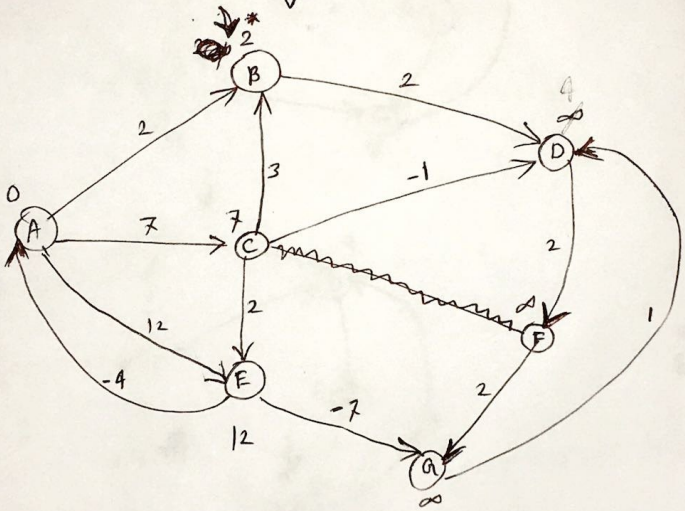
2018. 6 (a)

2018 6 (a) i)

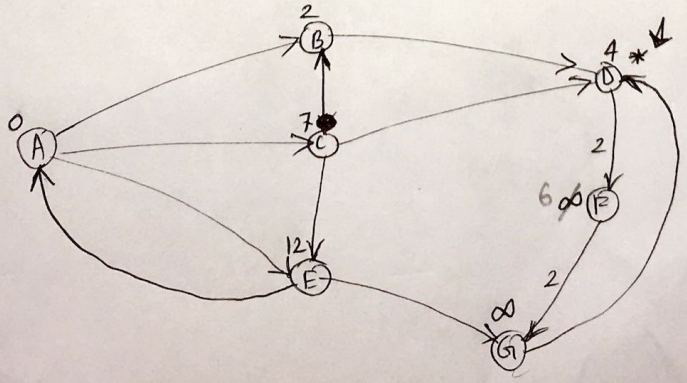


2 7 12
B, C, E

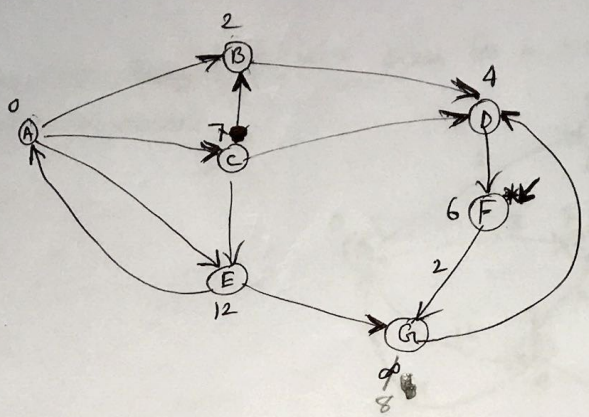
Elements in priority Q



7 12 4
C, E, D

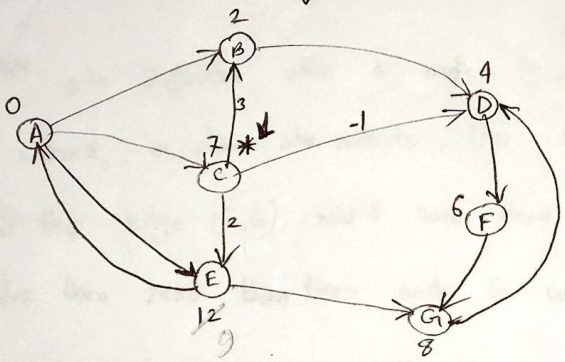


7 12 6
C, E, F

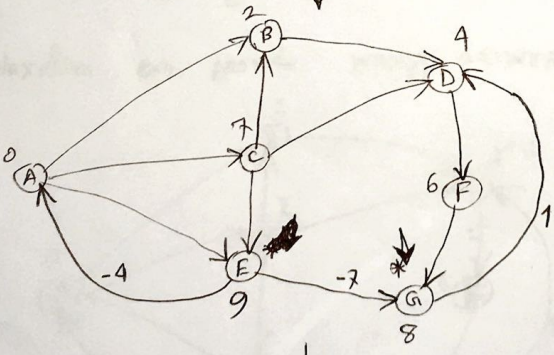


7 12 8
C, E, G

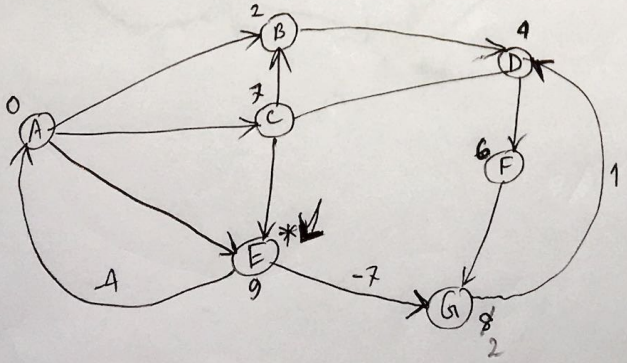
Elements in Priority Q



12 8 9
E, G, E



9 12
E, E



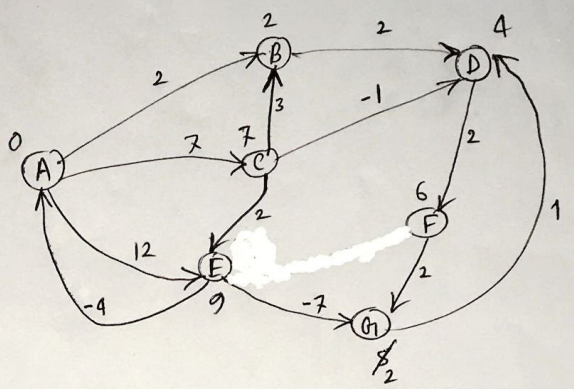
12 2
E, G

~~Now, important thing~~
~~Now, pause a moment~~

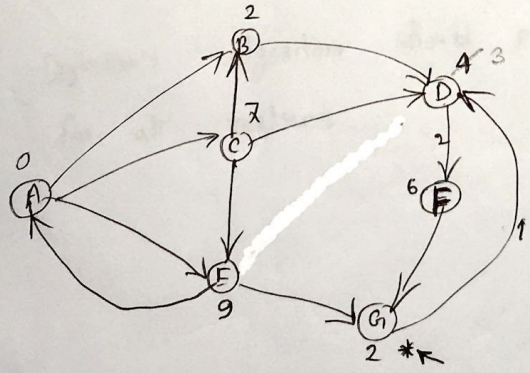
Now, let's pause for a moment.

In priority queue

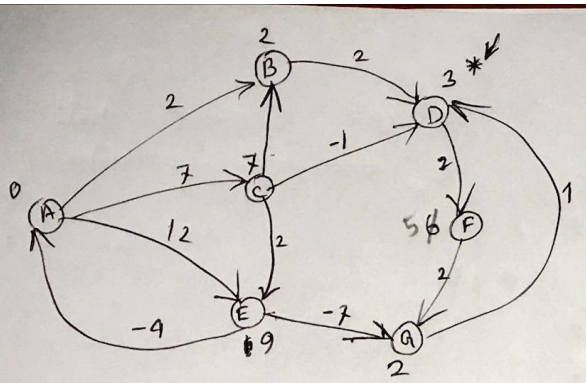
12 2
E , G



We know, In Dijkstra once a node is out of the priority queue, the node is closed, in other words, its shortest path has been found. If the edge (E,G) would have been any positive value greater than zero, then node G would not have been updated. After updating node G through (E,G) edge, any subsequent relaxation can provide wrong answer. Let's continue,



12 3
E , D



12 5
E , F ,

(iv) Nothing to relax .

i)

Node	A	B	C	D	E	F	G
Shortest path	0	2	7	3	9	5	2

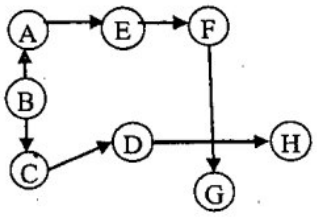
ii) For vertices D, F, G we got wrong path.

Shortest path of D, F, G ~~are~~ from A are 3, 5, 2 respectively.

The correct path for D, F, G should have been 4, 6, 8 respectively.

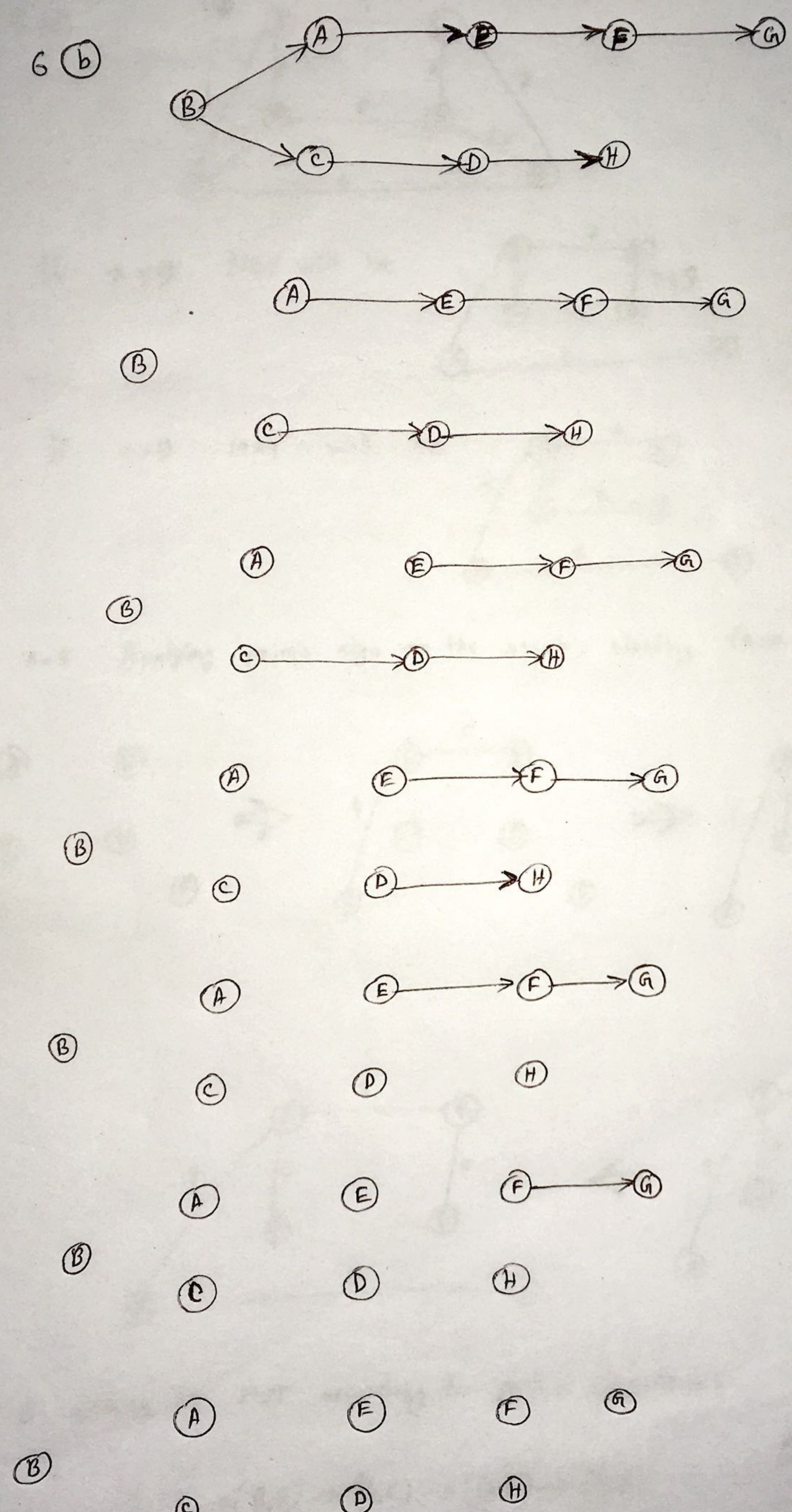
iii) If we remove the edge (E, G) with ~~the~~ weight -7, then Dijkstra's algorithm should provide correct ~~s~~ answers for all vertices.

b) Apply Topological sort in Lexicographical order of the following graph. If the graph has multiple answer then mention all of them. (10)



2018. 6 (b)

2018 6 (b)



Lexicographical Ordering: B A C D E F G H

c) Describe in steps how the Ford-Fulkerson algorithm finds the maximum flow. (05)

2018. 6 (c)

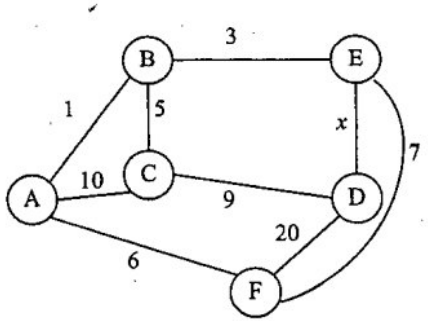
Ford-Fulkerson Algorithm

The algorithm follows:

1. Initialize the flow in all the edges to 0.
2. While there is an augmenting path between the source and the sink, add this path to the flow.
3. Update the residual graph.

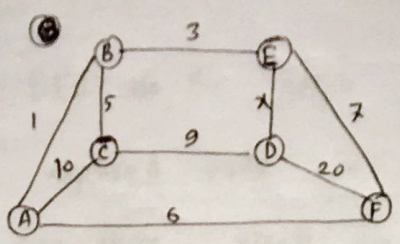
We can also consider reverse-path if required because if we do not consider them, we may never find a maximum flow.

7. a) For the following graph the bold edges form a minimum spanning tree what can you tell about the range of values for X? If $x = 5$, then use prim's algorithm starting at node A to compute the MST of the following graph. In particular write down the edges of the MST in the order in which prim's algorithm adds them to the MST. Use the format (node1, node2) to denote on edge. (12)

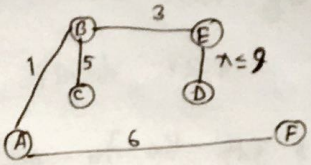


2018. 7 (a)

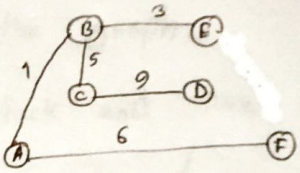
2018 7 (a)



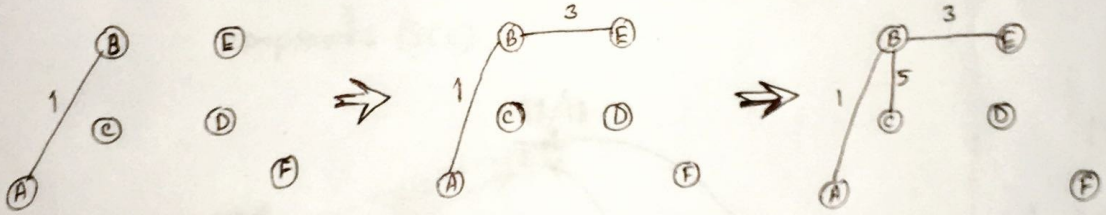
If $x \leq 9$ MST will be

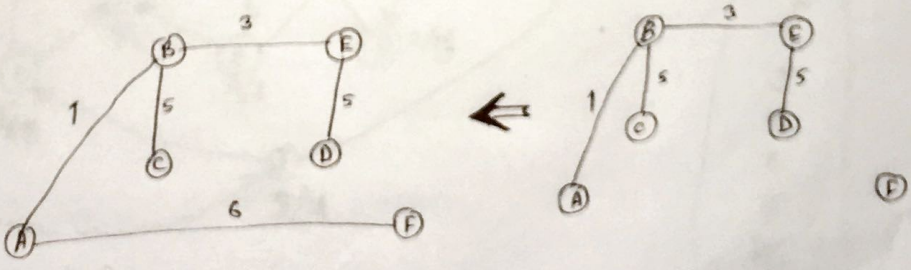


If $x > 9$ MST will be



If $x = 5$ Applying prim's algo on the graph, starting from A

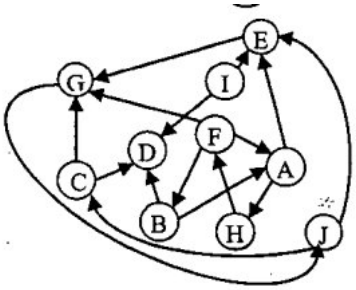




Orden of adding to MST according to prims algorithm:

$(A,B) \rightarrow (B,E) \rightarrow (B,C) \rightarrow (D,E) \rightarrow (A,F)$

b) What are the strongly connected components of the graph below? (13)



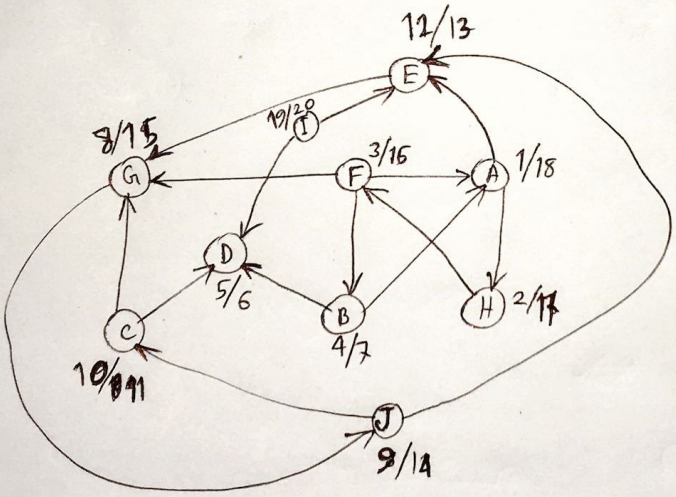
2018. 7 (b)

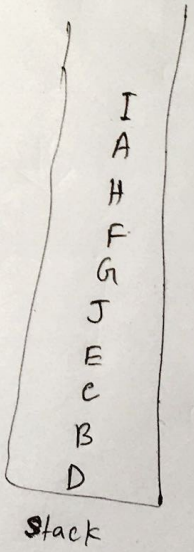
(2018) 7(b)

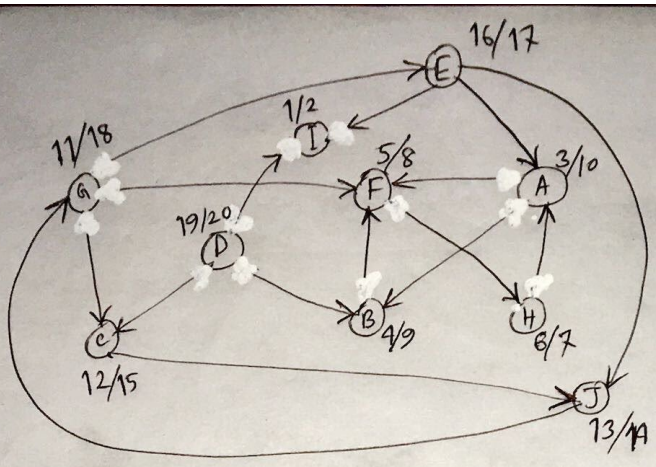
Apply DFS on the graph and store ~~every~~ any completely explored node in a stack. This way when we pop from stack we shall get ~~ete~~ nodes according to highest finish time.

Now reverse the direction of all the ~~the~~ & edges of the graph i.e transpose the graph.

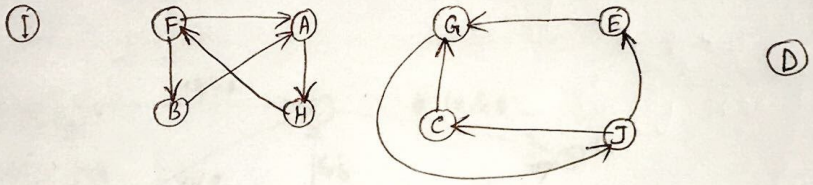
Then, pop a node from stack and run dfs on the reversed graph. The number of unique DFS calls will be the number of ~~scc~~ strongly connected components (SCC).



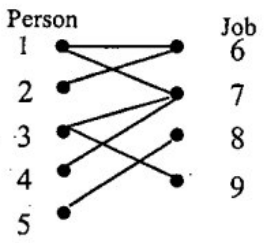




Strongly connected components: {I}, {A B F H}, {G C J E}, {D}



Define bipartite graph. Find the maximum matching using khokho idea of the following bipartite graph. (10)

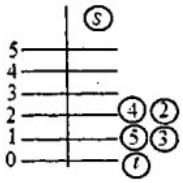


2018. 7 (c)

in Photo

couldn’t solve :(

8. a) Consider the following max flow problem given by the push relable algorithm. The distance labels are given at left the numbers above or below the nodes are accesses The numbers on the arcs are the residual capacities. (18)

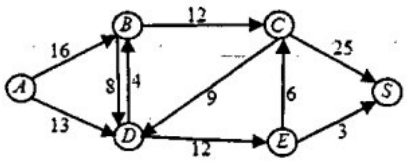


i) Is the labeling of the nodes valid? (ii) What are the admissible arcs? (iii)Which nodes are active? (iv)A potential method was used to prove the $O(V^2E)$ time bound-what was the potential method? Apply the potential method to calculate the max flow of the following network.

2018. 8 (a)

couldn’t solve sorry :(

b) Write down the differences between Ford – Fulkerson and Edmonds- Karp algorithm. Apply ford-Fulkerson algorithm for the network in following figure. (17)



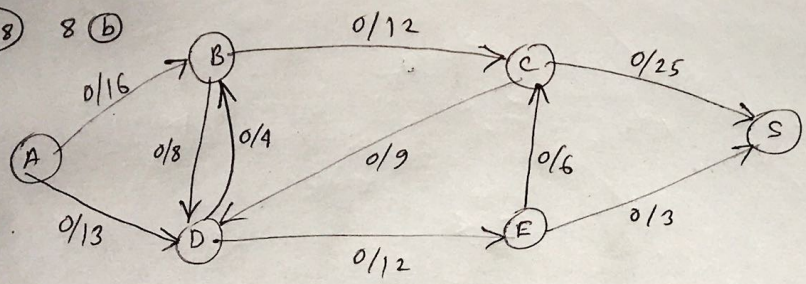
2018. 8 (b)

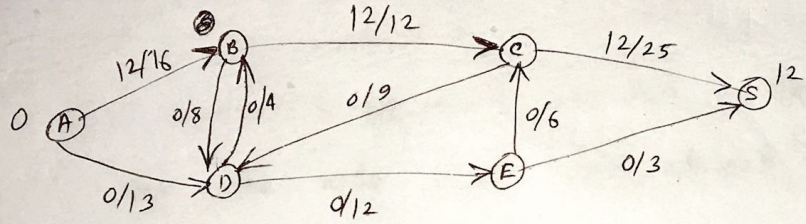
solution: The Edmonds-Karp Algorithm is a specific implementation of the Ford-Fulkerson algorithm. Like Ford-Fulkerson, Edmonds-Karp is also an algorithm that deals with the max-flow min-cut problem.

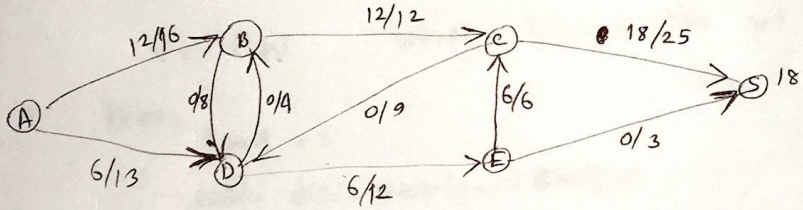
Ford-Fulkerson is sometimes called a method because some parts of its protocol are left unspecified. Edmonds-Karp, on the other hand, provides a full specification. Most importantly, it specifies that breadth first search should be used to find the shortest paths during the intermediate stages of the program.

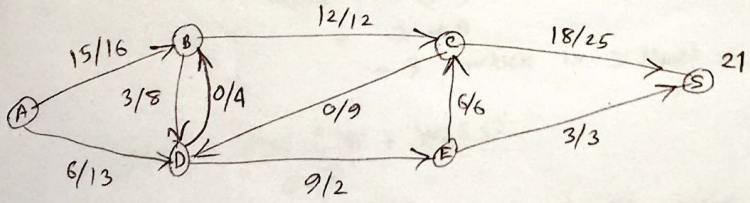
Edmonds-Karp improves the runtime of Ford-Fulkerson, which is $O(|E| \cdot f_*)$, to $O(|V| \cdot |E|2)$. This improvement is important because it makes the runtime of Edmonds-Karp independent of the maximum flow of the network, $f*$.

(2018) 8 (b)









Maximum Flow = 21