KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY
B.Sc. Engineering 2nd Year 2nd Term Examination, 2019
Department of Computer Science and Engineering
CSE 2201
Algorithm Analysis and Design

TIME: 3 hours FULL MARKS: 210

N.B. i) Answer **ANY THREE** questions from each section in separate scripts.
ii) Figures in the right margin indicate full marks.

SECTION A

(Answer **ANY THREE** questions from this section in Script A)

1. a) What are the differences between performance analysis and performance measurement of an algorithm. (07)
   b) Consider a 0/1 Knapsack problem. Explain the application of dynamic programming and greedy algorithm to find an optimal solution. Give the two separate repetition of problem space and compare the time complexity of the algorithm under the said paradigm. (11)
   c) "Is the minimum spanning tree generated using both Krushkal's and Prim's unique" Explain your answer if you say 'yes' or give a counter example if you say 'no'. (09)
   d) Write the control abstraction for greedy method. (08)

Question Solution – CSE 2201 (2019)

1809031
Waliul

1

a) Ans:

Performance analysis estimates space and time complexity in advance. We usually use asymptotic notations to do so.

There are mainly three asymptotic notations: Big-O notation, Omega notation and Theta notation. We consider the input value to be very large.

On the other hand, performance measurement measures the space and time ~~or~~ taken in actual runs i.e. we count the space and time in runtime. For measuring time, a stopwatch can be used.

b) Ans:

In a greedy algorithm, we always tend to pick an object that will make our profit more than the other objects. As this is a 0/1 knapsack problem, we can not take any fraction of an object. So in greedy method, at first we sort the objects in respect to their profit in descending order.

Then we pick the objects from the very front until the total weight of our knapsack becomes equal or less than the constraint weight.

If the value of x_i is 1 then we take the i^{th} object, 0 otherwise. ith object has P_i profit and W_i weight. The capacity of knapsack is M.

In greedy method, we may find a feasible solution, but that will not be optimal solution.

In dynamic programming, we either take an object, or don't. And as we solve the same sub-problems again and again, we can store the solution to the subproblems in a 2D array dp[n][w], where n is the number of elements and w is the maximum capacity. The state DP[i][j] will denote maximum value of 'j-weight' considering all values from '1 to ith'.

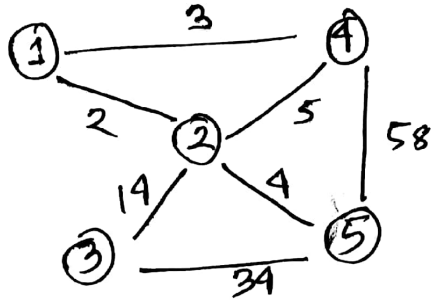
In greedy method, first we sort the array and then we take the objects : so, time complexity would be $O(n \log n)$, where n is the total objects.

In dynamic problem, we use recursion for nth depth ~~and~~ return when we placed the maximum capacity. So, the time complexity would be $O(n \cdot W)$, where n is the number of objects and W is the capacity of the knapsack.

c) Ans:

No, the minimum spanning tree generated using both kruskal's and Prim's is not unique.

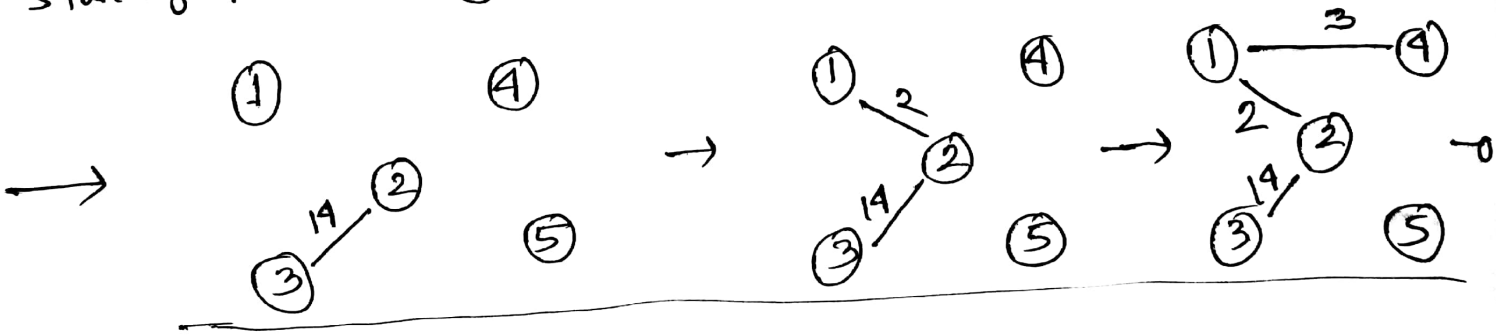
Example:



Generating the MST using Prim's and Kruskal's.

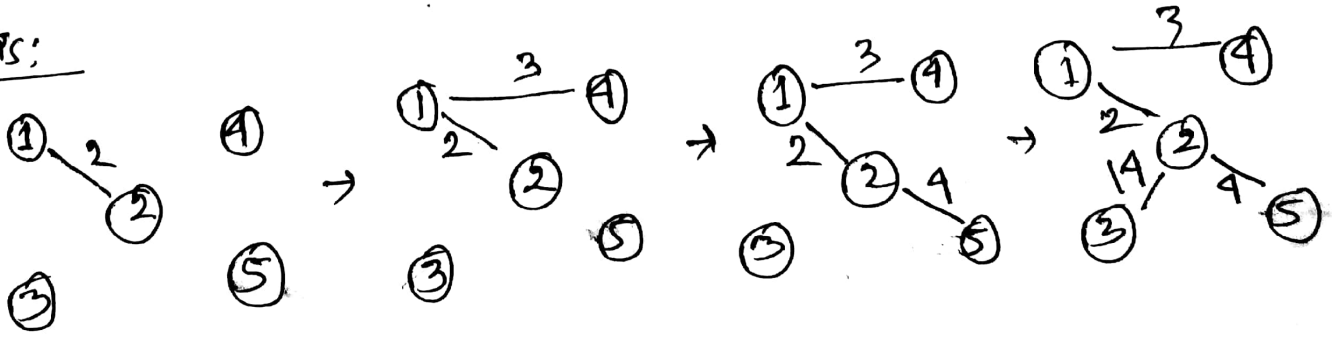
Prim's:

Starting from node (3)



(all vertices are covered)

Kruskal's:



As we can see, the mst's are the same, when the cost of edges are unique.

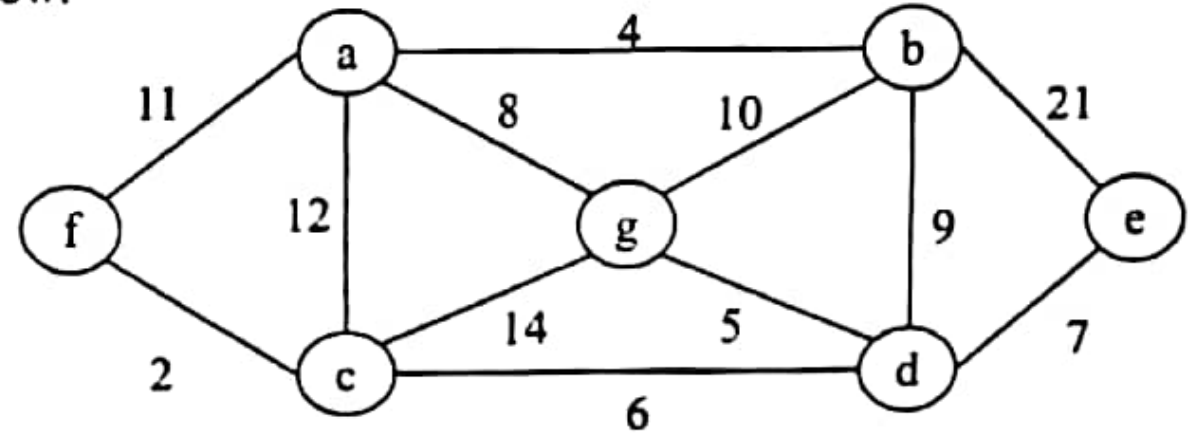
d) Ans:

Control Abstraction for greedy method:

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

2. a) Define BFS and DFS. What are the time complexity of BFS and DFS of a graph? (06)

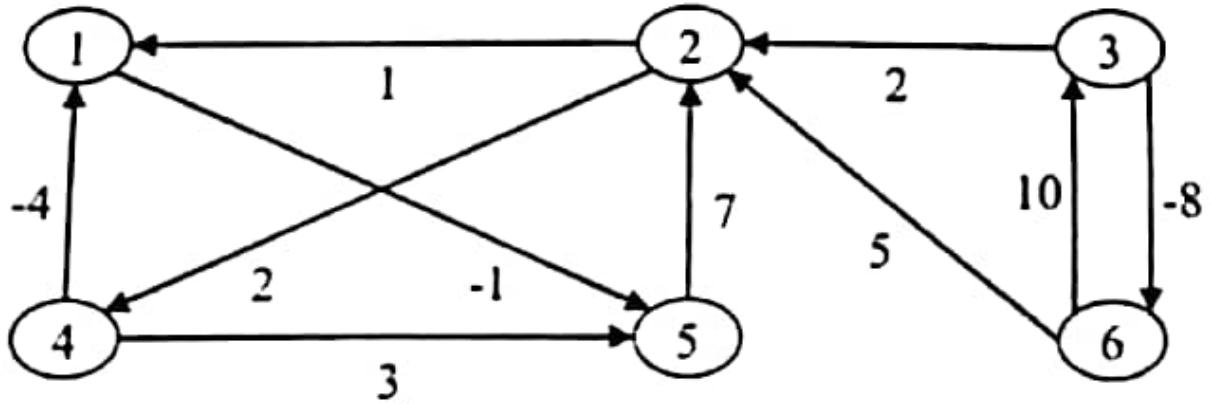
b) What is Spanning Tree? Use Prim's algorithm to determine the Minimum Cost Spanning Tree of the graph below. (10)



What is the total cost of the tree?

c) Use example to distinguish between feasible solution and optimal solution for the case of *Knapsack problem* (07)

d) Run the Floyd-Warshall algorithm on the weighted directed graph shown in the following figure. (12)



Show the Matrix (All pair) D^k that results for each iteration of the outer loop.

2]

a) Ans:

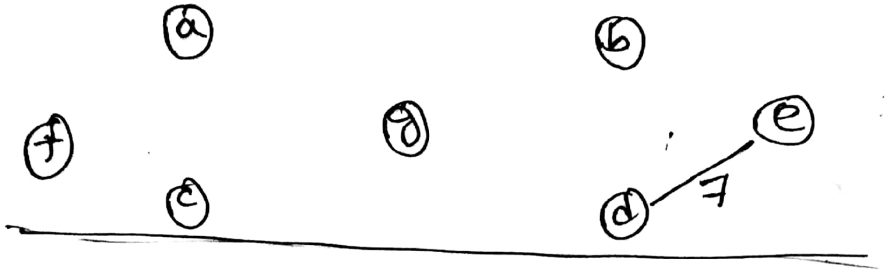
BFS (Breadth First Search) is a vertex based technique for finding a shortest Path in graph. It uses a Queue data structure which follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in queue.

DFS (Depth First Search) is an edge based technique. It uses the Stack data structure, performs two stages, first visited vertices are pushed into stack and second if there is no vertices then visited vertices are popped.

Considering V is the number of ~~edg~~ vertices and E is the number of edges, the time complexity of both BFS and DFS is $O(V+E)$.

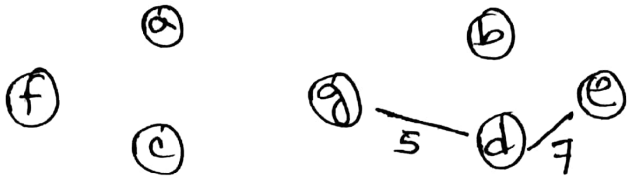
b) Ans:

A spanning tree is a subset of graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree doesnot have cycles and it cannot be disconnected.

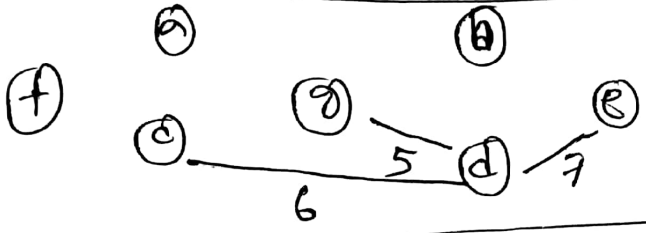


[starting from e]

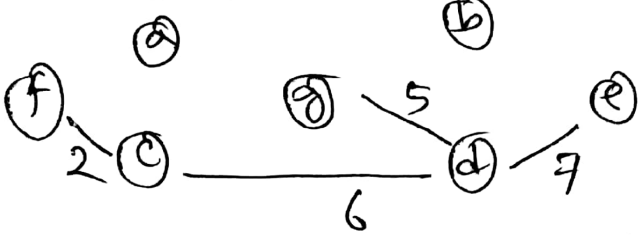
[step-1]



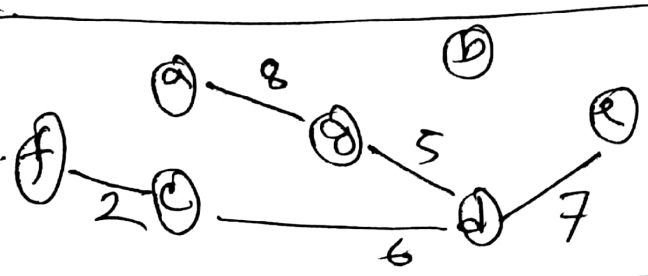
[step-2]



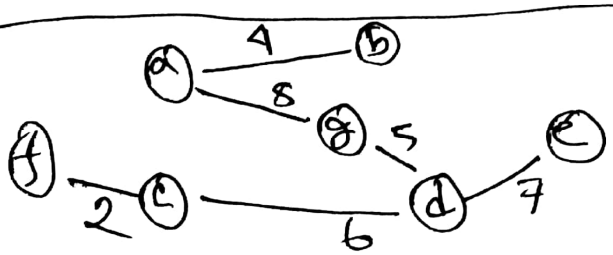
[step-3]



[step-4]



[step-5]



∴ the total cost of the tree = 4+8+5+6+2+7 = 32

c) Ans:

$P_1, P_2 \cdots P_n$ (Profit)

$W_1, W_2, \cdots W_n$ (weight)

M (capacity)

The goal is, maximize $\sum_{i=1}^{n} P_i x_i$, subject to $\sum_{i=1}^{n} W_i x_i \le M$, $x_i = 0$ or 1

or minimize $-\left(\sum_{i=1}^{n} P_i x_i\right) = Y$

The upper bound can be calculated by quickly finding a feasible solution; starting from the smallest available i, scanning towards the largest i's until M is exceeded.

Example:

$n=6$, $M=34$

i	1	2	3	4	5	6
P_i	6	10	4	5	6	4
W_i	10	19	8	10	12	8

(sorted by P_i/W_i ratio)

$P_i/W_i \ge P_{i+1}/W_{i+1}$

A feasible solution: $X_1=1$, $X_2=1$, $X_3=0$, $X_4=0$, $X_5=0$, $X_6=0$

$\therefore -(P_1+P_2) = -16$ (Upper bound)

Any solution higher than -16 can not be an optimal solution.

For fractional, $Y' = -\sum_{i=1}^{n} P_i x_i'$, where $0 \le x_i \le 1$, and this is an optimal solution for knapsack problem.

$\therefore Y' \le Y$ (they are negative)

For fractional, $X_1=1$, $X_2=1$, $X_3 = \frac{(34-10-19)}{8} = \frac{5}{8}$, $X_4=0$, $X_5=0$, $X_6=0$

$\therefore -(P_1+P_2+5/8\,P_3) = -18.5$ (lower bound) ≈ -18 (only consider integer)

$\therefore$ So, the optimal solution will be between 16 and 18.

d) Ans:

The corresponding matrix:

• bear with me guys *

Node	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	∞	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	3	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

Outer loop will iterate for 6 times. Let k be current node of outer loop.

After k=1 complete,

Node	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	-5	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

After k = 2 completion,

Node	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	3	2	0	4	2	∞ 8
4	-4	∞	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	5	0

After k = 3 completion,

Node	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	3	2	0	4	2	∞ 8
4	-4	∞	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	2	10	7	5	0

After k = 4 completion,

Node	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	-2	0	∞	2	-3	∞
3	-1	2	0	4	-1	∞ 8
4	-4	∞	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	2	10	7	2	0

After k=5 Completion,

Node	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	0	2	0	4	-1	-8
4	-4	-2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	2	10	7	2	0

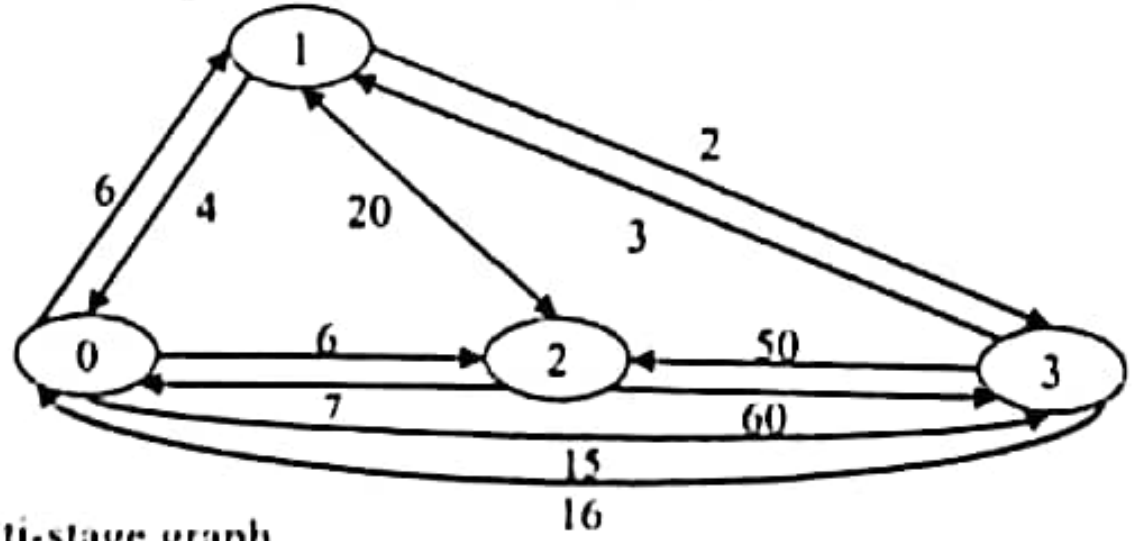
After k=6 completion,

Node	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-6	0	-1	-6	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	2	10	7	2	0

This is the final Matrix (All pair).

3. a) Define Implicit and Explicit constraints. Write Implicit and Explicit constraints for n-queens and Sum of Subset problems. (06)

b) Consider the following Travelling Salesman Problem (TSP): (14)



i) Convert Multi-stage graph.

ii) Find the minimum cost path in the Multi-stage graph (from (i)). Do this using the forward reasoning approach.

c) Apply backtracking technique to solve the following instance of Subset Sum problem: (10)

$S = \{1, 3, 4, 5, 8\}$ and $d = 16$.

d) What are the differences between Branch-and-Bound and back tracking paradigm? (05)

3

a) Ans:

Implicit Constraints: The implicit constraints are rules that determine which of the tuples in the solution space of I satisfy the criterion function. Thus implicit constraints describe the way in which the x_i must relate to each other.

Explicit Constraints: Explicit constraints are rules that restrict each x_i to take on values from a given set

N-queens:

Let us number the rows and columns of the chessboard 1 through N. The queens can also be numbered 1 through N. Since each queen must be on a different row, we can, without loss of generality, assume queen i to be placed on row i. All solutions to the N-queens problem can therefore be represented as N-tuples $(x_1, x_2, \ldots x_n)$, where x_i is the column on which queen i is placed.

The explicit constraints using this formulation are $S_i = \{1, 2, 3, \ldots N\}$, $1 \le i \le N$.

The implicit constraints for this problem are that no two x_i's can be the same (i.e. all queens must be on different columns) and no two queens can be on the same diagonal.

Sum of subset:

Given positive numbers w_i, $1 \le i \le n$, and m, which is the sum of subset.

The explicit constraints require $x_i \in \{j \mid j \text{ is an integer and } 1 \le j \le n\}$.

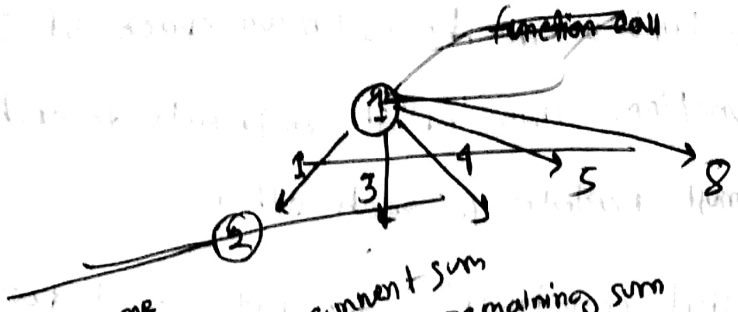
The implicit constraints require that no two be the same and that the sum of the corresponding w_i's be m. Another implicit constraint, $x_i < x_{i+1}$, $1 \le i < k$

where x_i is the indice.

b) Ans:

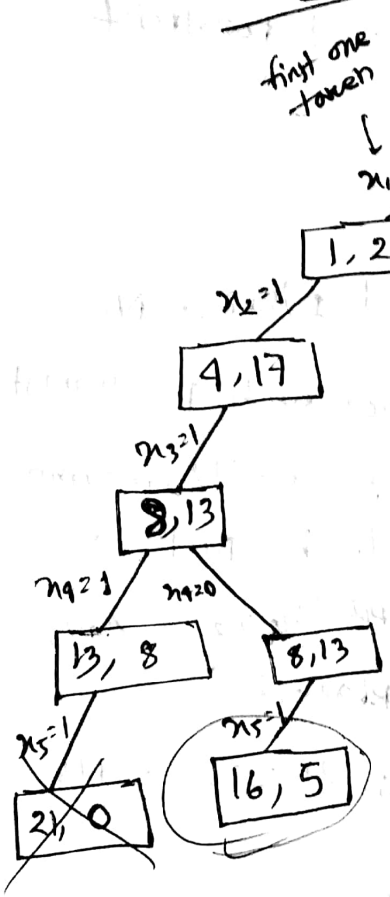
* সাধা চুনিয়া সুইজলা solution পাই না *

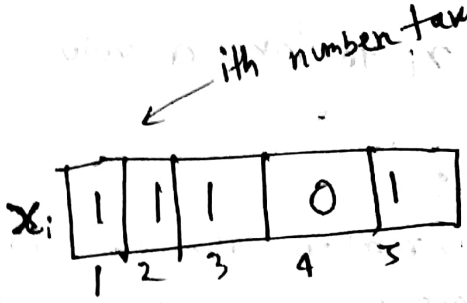
c) Ans:



$S = \{1, 3, 4, 5, 8\}$

$d = 16$





d) Ans:

Branch-and-Bound is used to solve optimization problems. When it realises that it already has a better optimal solution that the pre-solution leads to, it abandons that pre-solution. It completely searches the state space tree to get optimal solution. It traverses the tree in any manner, DFS or BFS.

Backtracking is used for solving Decision Problem. It is used to find all possible solutions available to a problem. When it realises that it made a bad choice, it undoes the last choice by backing it up. It searches the state space tree until it has found a solution for the problem. It uses DFS to traverse the state space tree.

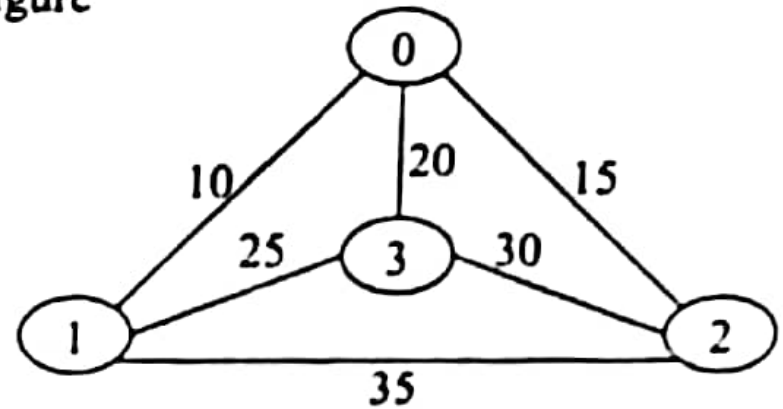
4. a) Draw the State-space tree for 4-queen problem. How is the solution reduced from 4^4 to optimal solution? (09)

b) Consider the following table: (13)

I	1	2	3	4	5
P_I	0.24	0.22	0.23	0.3	0.01

Construct OBST for keys with the given probabilities.

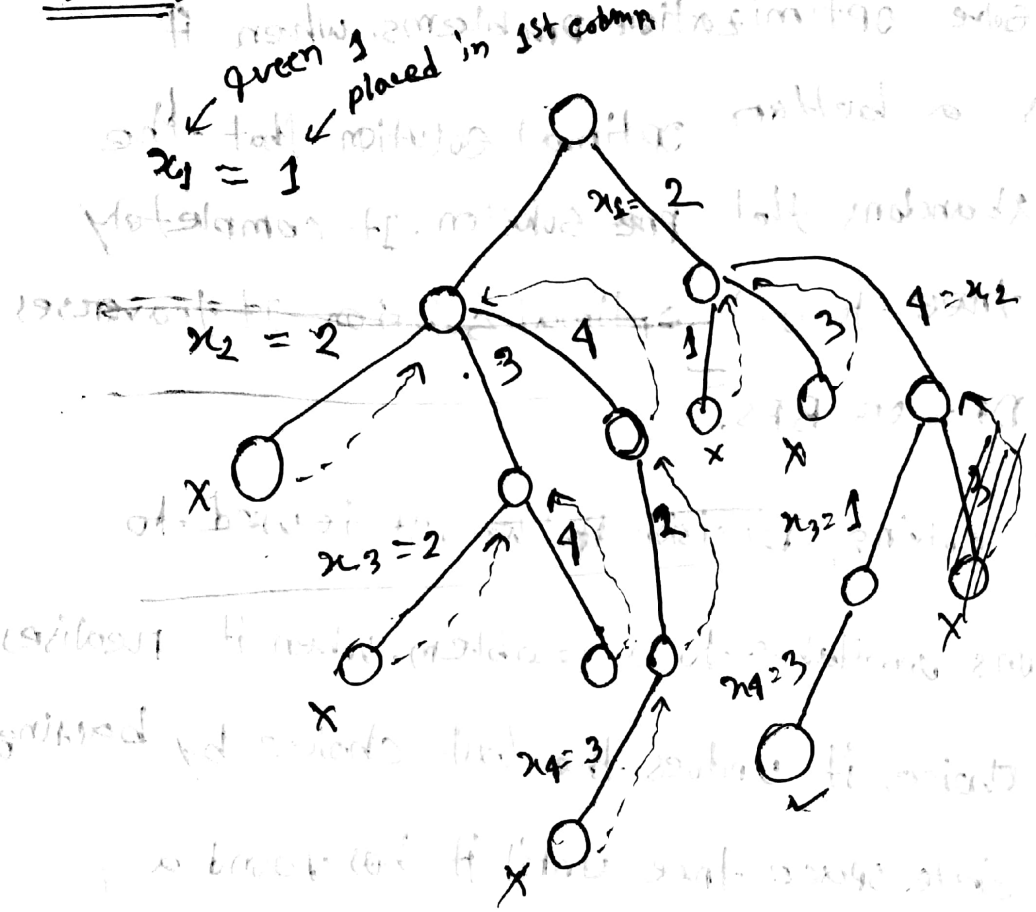
c) Consider the following figure (13)

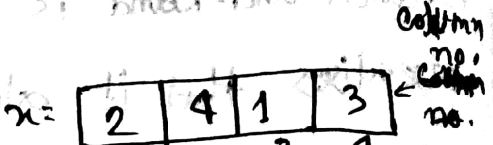


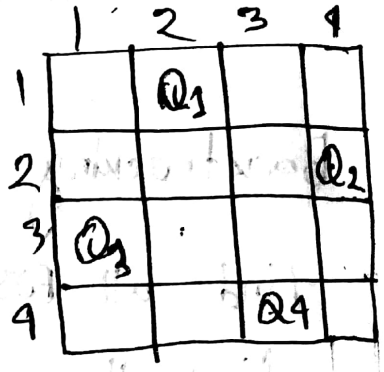
Find the optimal solution for TSP using Branch-and-Bound technique.

4]

a) Ans:

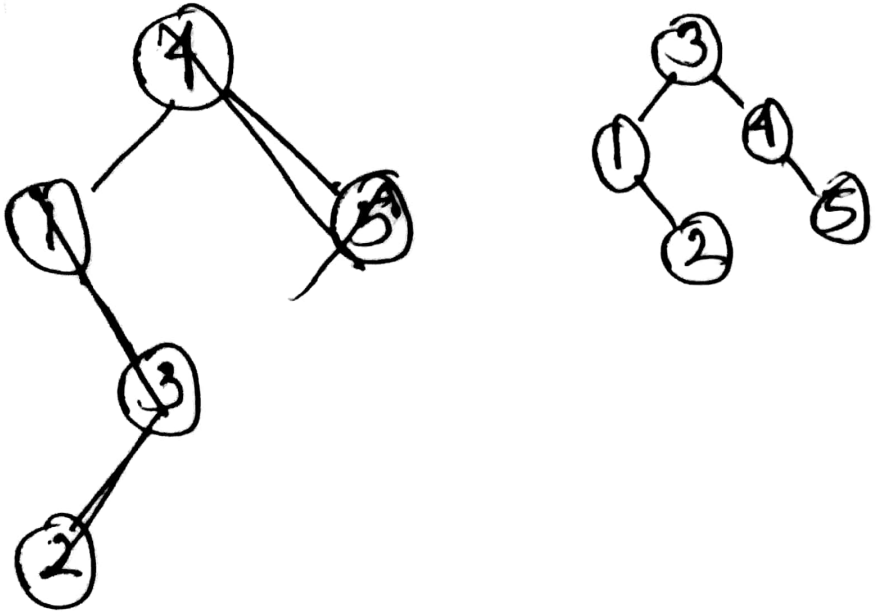




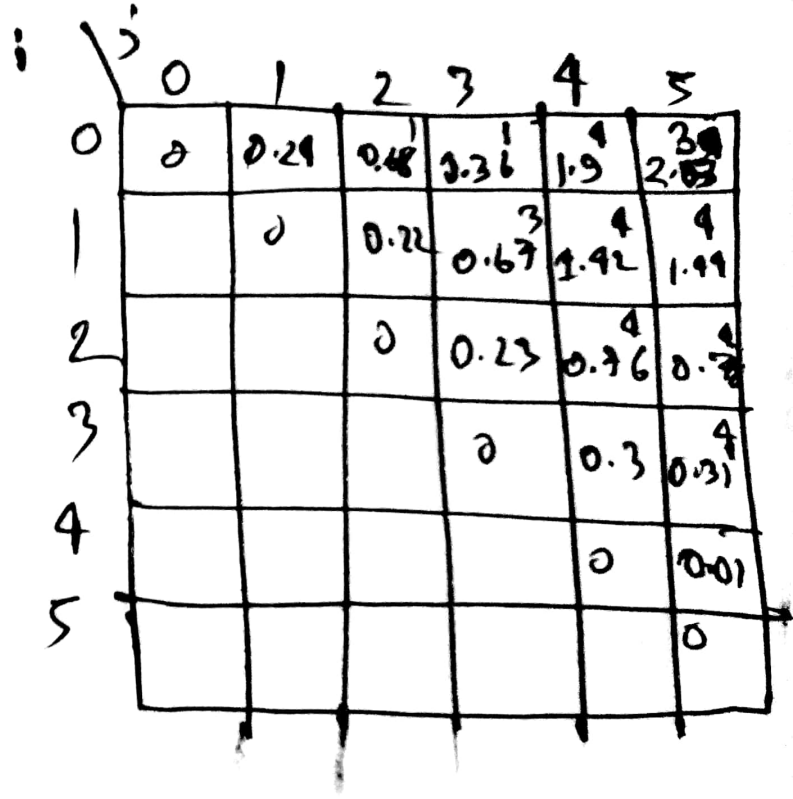


Using backtracking, the solution is reduced from 4^4 to optimal solution.

b) Ans:



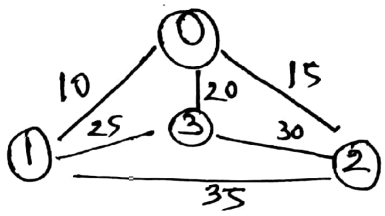
i	1	2	3	4	5
Pi	0.29	0.22	0.23	0.3	0.01



This is the optimal binary search trees.

(Watch Abdul Bari's video for understanding)

(c) Ans:



	0	1	2	3	
0	∞	0	5	10	10
1	0	∞	25	15	10
2	0	20	∞	15	15
3	0	5	10	∞	20
					55

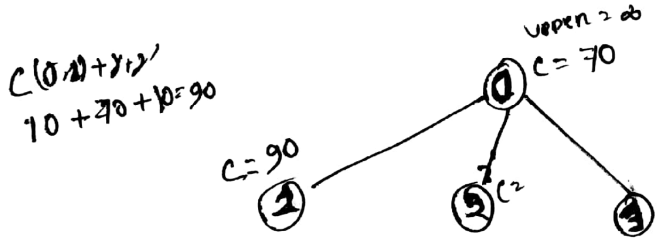
	0	1	2	3	
0	∞	10	15	20	10
1	10	∞	35	25	10
2	15	35	∞	30	15
3	20	25	30	∞	20

	0	1	2	3	
0	∞	0	0	0	10
1	0	∞	20	5	10
2	0	20	∞	5	15
3	0	5	5	∞	20
	0	0	5	10	15+55 = 70

← This is reduced matrix

← reduced cost

← atleast the cost will be 70



[I'm sorry, I couldn't do it ~~any~~ further more

Watch Abdul Bari's video if you want to solve it. (Travelling Salesman Problem - Branch and Bound)

(The solution may be: 0 → 1 → 3 → 2 → 0, cost = 80)

5. a) What do you mean by algorithm? Write down the basic characteristics of an algorithm. (10)
   b) Define running time of an algorithm. Discuss some running time functions of an algorithm. (10)
   c) Consider the code segments given in the following figure and calculate time frequency of each of the segments. (15)

i)
```
for(i=0;i<n;i++)
    {
 for(j=0;j<n;j=j*2)
    {stmnt;}
    }
```

ii)
```
p = 0;
for(i=1;i<n;i*2)
      { p++;}
for(j=1;j<p;j=j*2)
      {stmnt;}
```

iii)
```
i = 1; k = 1;
 while(k < n){
   stmnt;
   k =k+1;
   i++;}
```

5]

a) Ans:

An algorithm is a set of instructions for solving a problem or accomplishing a task.

@ The basic characteristics of an algorithm:

Input: An algorithm should have 0 or more well-defined inputs.

Output: An algorithm should have atleast 1 output

Definiteness: Each instruction should be clear.

Finiteness: Algorithm must terminate after a finite number of steps.

Effectiveness: Instruction must be basic to carry out principle.

b) Ans:

The running time of an algorithm is the length of time taken to run the program on some standard computer.

Θ-notation:

For a given function $g(n)$, we denote $\Theta(g(n))$ the set of functions,

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2 and n_0 such that

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.

O-notation:

For a given function $g(n)$, we denote $O(g(n))$ the set of functions,

$O(g(n)) = \{f(n)$: there exist positive constants C and n_0 such that

$0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$

Ω-notation:

$\Omega(g(n)) = \{f(n)$: there exists +ve constants c and n_0, $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$

c) Ans:

i)
```
For(i=0; i<n; i++)        ⟶ n+1
{
  For(j=0; j<n; j=j*2)    ⟶ n × ∞
  {
    stmnt;                ⟶ n × ∞
  }
}
```

$T(n) = n+1+n\times\infty+n\infty$

ii)
```
P=0;                      ⟶ 1
For(i=1; i<n; i*2)        ⟶ ∞ (infinity)
  { P++;                  ⟶ ∞
  }
For(j=1; j<P; j=j*2)      ⟶ ∞
  { stmnt;                ⟶ ∞
  }
```

$T(n) = 4\infty$

iii)
```
i=1;                      ⟶ 1
k=1;                      ⟶ 1
while(k<n)                ⟶ k
{
  stmnt;                  ⟶ k
  k=k+i;                  ⟶ k
  i++;                    ⟶ k
}
```

$\therefore T(n) = 4k+2$

$= 4\cdot\left(1+\frac{p(p+1)}{2}\right)$

$= 4+2p^2+2p$

(assume $k>n$)

i	k
1	1+1 = 2
2	1+1+2 = 4
3	1+1+2+3 = 7
⋮	⋮
p	1+1+2+...+p

$\therefore k = 1+\frac{p(p+1)}{2}$

$\Rightarrow 1+\frac{p(p+1)}{2} > n$

$\Rightarrow 1+\frac{p^2+p}{2} > n$

$\Rightarrow p^2 > n \Rightarrow p > \sqrt{n}$

$\therefore O(\sqrt{n})$

6. a) What do you mean by recurrence? What are the methods to solve the recurrence? Solve the following recurrences: *(10)*

   i) $T(n) = 2T(\sqrt{n}) + \log(n)$

   ii) $T(n) = T(n-a) + T(a) + n$

   b) Give the best Big-Oh characterization for each of the following running time estimates (where n is the size of the input problem) *(15)*

   i) $(n+1)^3$ ii) $\sum_{i=1}^{n} i$ iii) $n!$ iv) $6n^3/(\log n + 1)$ v) $6 * 2^n + n^2$

   c) Define algebraic simplification. Explain the principal of algebraic simplification using dense polynomial representation. *(10)*

6]

a) Ans:

A recurrence is an equation or inequity that describes a function in terms of its value on smaller inputs.

There are four methods for solving recurrence:

1. Substitution method
2. Iteration method
3. Recursion Tree Method
4. Master Method

i) $T(n) = 2T(\sqrt{n}) + \log n$ ——— (1)

$T(n) = 2[2T(\sqrt{\sqrt{n}}) + \log(\sqrt{n})] + \log n$

$T(n) = 4T(n^{\frac{1}{4}}) + 2\log(n^{\frac{1}{2}}) + \log n$ ——— (2)

$T(n) = 4[2T(n^{\frac{1}{8}}) + \log(n^{\frac{1}{4}})] + 2\log(n^{1/2}) + \log n$

$T(n) = 8T(n^{1/8}) + 4\log(n^{1/4}) + 2\log(n^{1/2}) + \log n$ ——(3)

⋮ (k times)

$T(n) = 2^k \cdot T(n^{1/k}) + \log n + 2\log(n^{1/2}) + \cdots + 2^{k-1}\log(n^{1/k})$

$\therefore$ ~~$n-k=0$~~ $n=k$ [assume $T(n)=1$ when $n=0$]

[$n^{1/k} = 1 \Rightarrow \frac{1}{k}\log n = \log 1 \Rightarrow \frac{\log n}{\log 1} = k$

~~$T(n) = 2^n T(n^{1/n}) + \log n + 2\log(n^{1/2}) + \cdots + 2^{n-1}\log(n^{1/n})$~~

[I don't know what to do any more.]

b) Ans :

i) $(n+1)^3$

Definition of Big-oh suggests

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

where $O(g(n))$ would be the Big-oh characterization of the function $f(n)$.

$$n^3 + 3n^2 + 3n + 1 \leq n^3 + 3n^3 + 3n^3 + 1$$

$$\Rightarrow n^3 + 3n^2 + 3n + 1 \leq n^3 (1 + 3 + 3 + 1)$$

$$\Rightarrow n^3 + 3n^2 + 3n + 1 \leq \frac{8n^3}{g(n)}$$

$$\therefore f(n) = O(g(n)) = O(n^3)$$

ii) $\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + (n-1) + n$

$$= \frac{n(n+1)}{2}$$

Here,

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

$$\Rightarrow 1 + 2 + \cdots + (n-1) + n \leq n + n + \cdots + n$$

$$\Rightarrow \frac{n(n+1)}{2} \leq n^2$$

$$\Rightarrow \frac{n^2}{2} + \frac{n}{2} \leq n^2 + n^2$$

$$\Rightarrow \underbrace{\frac{n^2}{2} + \frac{n}{2}}_{f(n)} \leq \frac{2}{c} \cdot \frac{n^2}{g(n)}$$

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

iii) $n!$

iv) $6n^3/(\log n + 1)$

v) $6 * 2^n + n^2$

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

$$6 * 2^n + n^2 \leq 6 * 2^n + 2^n$$

$$\Rightarrow 6 * 2^n + n^2 \leq 2^n (6+1)$$

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

Q) Ans:

Algebraic Simplification is the process of writing an expression in the most efficient and compact ~~form~~ without affecting the value of the original expression.

Let's consider a polynomial. A univariate polynomial is generally written as

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

where x is an indeterminate and the a_i may be integers, floating point numbers, ~~or~~ When considering the representation of a polynomial by its coefficient, there ~~are at least~~ are at least two alternatives. The first calls for storing the degree ~~followed~~ by degree+1 coefficient; (Degree be n, $a_n \neq 0$)

$$(n, a_n, a_{n-1}, \ldots, a_1, a_0)$$

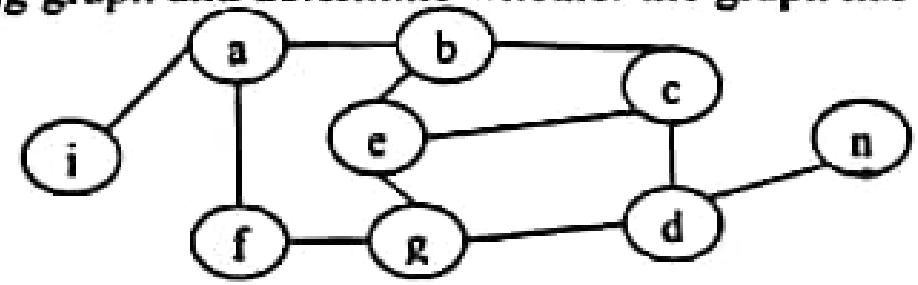
This is termed dense representation because it explicitly stores all coefficient whether or not they are ~~zero~~.

7. a) Let $C(x) = A(x) \times B(x)$, where $A(x) = 3x^2 + 4x + 1$ and $B(x) = x^2 + 2x + 5$. Now, find the resultant polynomial $C(x)$ using algebraic transformation and evaluation. (10)

b) "Any algorithm that computes the largest and smallest elements of a set of n ordered elements requires $(\lceil 3n/2 \rceil - 2)$ comparison." – Prove the statement. (05)

c) What is tight lower bound? Show that the lower bound of an insertion sort is tight. (10)

d) Consider the following graph and determine whether the graph has a cycle using DFS. (10)



7|

a) Ans: [Didn't find solution anywhere]

b) Ans: [sorry]

c) Ans: [sorry]

d) Ans:

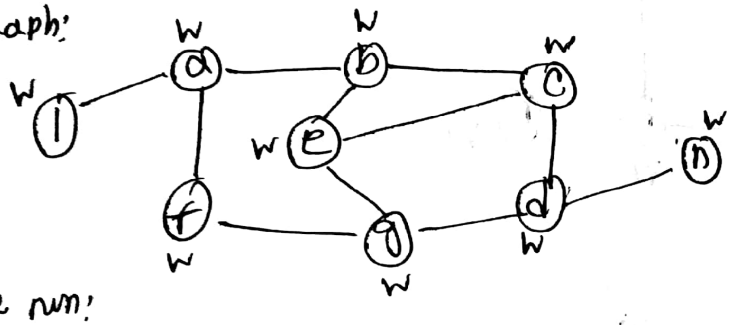
Lets consider somethings. If a node is

(W) white = not found/visited yet

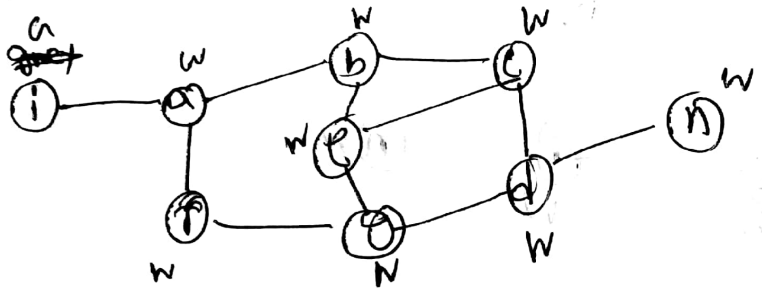
(G) grey = found but hasn't finished visiting its nodes yet

(B) black = finished working / visited.

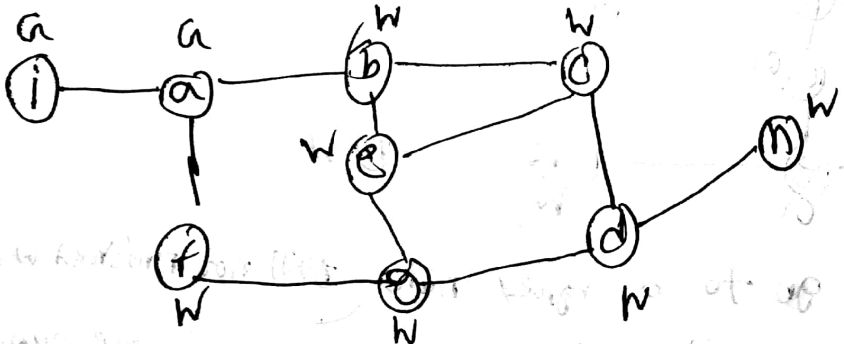
the graph:



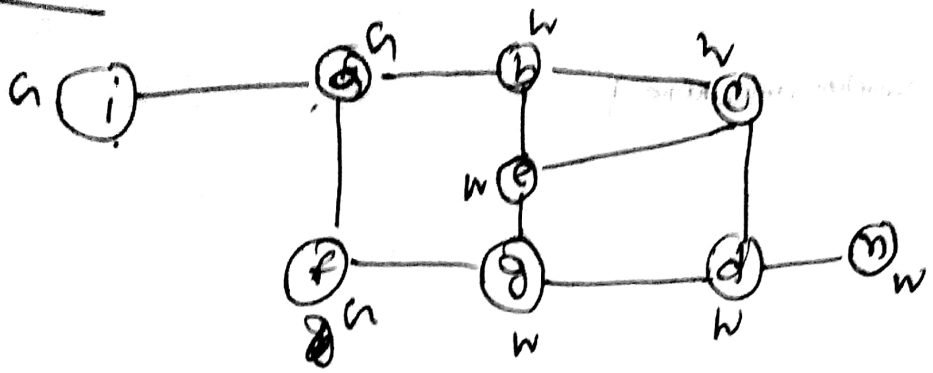
first ite run:



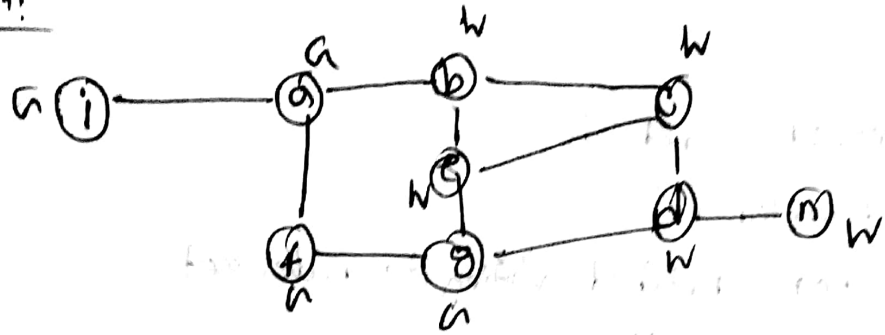
2nd ite



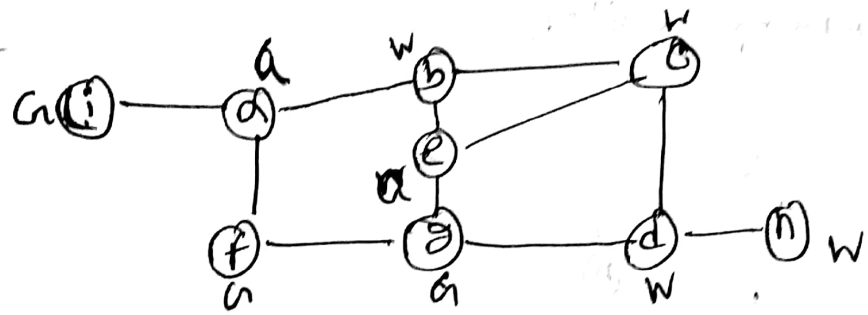
3rd:



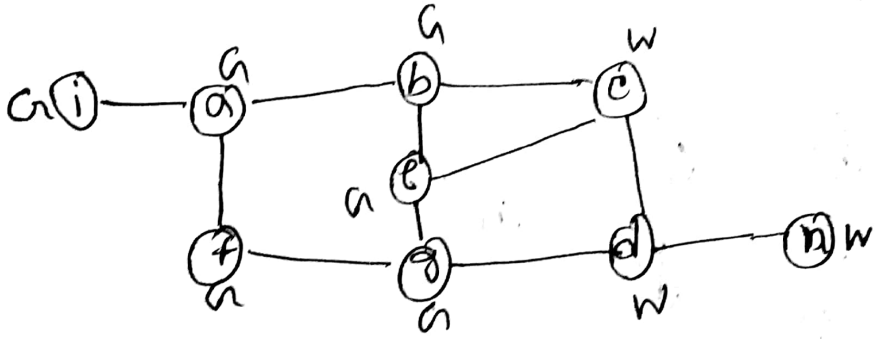
4th:



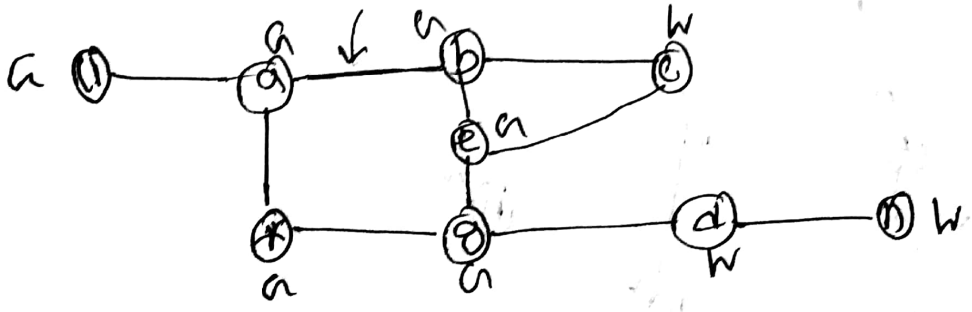
5th:



6th:



7th:



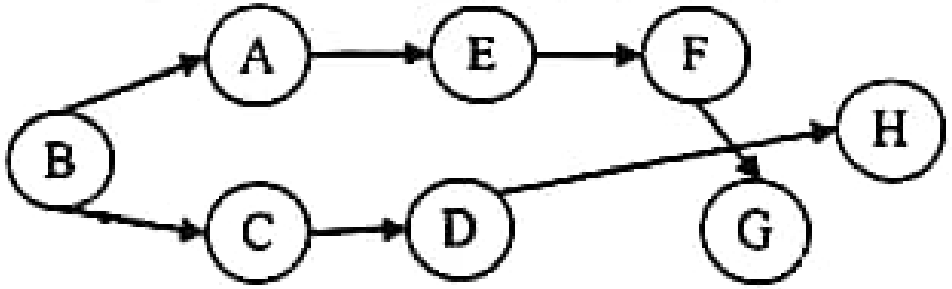
As we are attempting to go to a grey node (still not finished working) from a grey node (still working), there is a ~~cycle~~ at least one cycle and that is $a \to f \to g \to e \to b \to a$.

8. a) Why do we need non-deterministic algorithm? Convert a deterministic search into a non-deterministic search. (10)

b) Justify the statement – "the Halting Problem is a NP-hard problem that is not NP." (06)

c) Discuss the basic paradigm to find space complexity of a recursive algorithm. (10)

d) Apply Topological sort in Lexicographical order of the graph mentioned in the following figure. If the graph has multiple answer, then mention all of them. (09)



8|

a) Ans:

We need non-deterministic algorithm, for finding approximate solutions, when an exact solution is difficult on expensive to derive using a deterministic algorithm.

A nondeterministic search algorithm:

```
j := Choice(1,n);
if A[j] = x then { write(j); success(); }
write(0); Failure();
```

b) Ans:

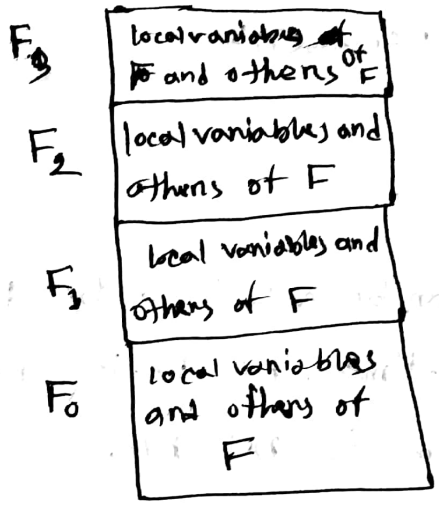
A problem will be NP if it can be verified by deterministic touring machine in polynomial time. As halting problem is un-decidable. So, Halting problem is not NP.

To show that Halting problem is NP-hard, we show that satisfiability is of halting problem. For this let us construct an algorithm A whose input is a prepositional formula X. Suppose X has n variables. Algorithm A tries out all 2^n possible truth assignments and verifies if X is satisfiable. if it is satisfied then A stops. If X is not satisfiable, then A enters an infinite loop. Hence A halts on input if X is satisfiable. If we had a polynomial time algorithm for the halting problem, then we could solve the satisfiability problem in polynomial time using A and X as input to the algorithm for the halting problem.

Hence, the halting problem is an NP-hand problem which is not NP.

(c) Ans:

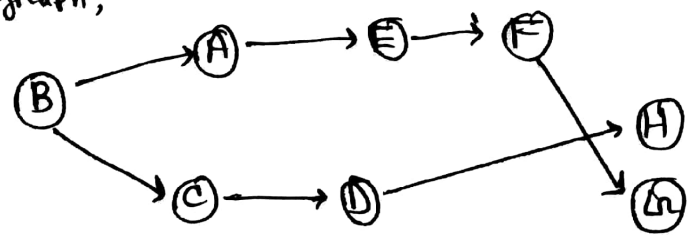
To find space complexity of a recursive algorithm, we have to consider the depth of the recursive call. And we have to calculate how much space it is taking in each call. If a function F is recursive, and its depth is N, then the stack looks like this,



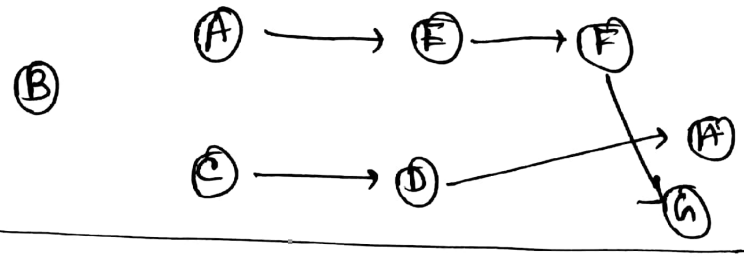
if N is 4, that is the depth of the recursive tree, and in each call, it F takes S(f) space, then the space complexity will be mostly NXS(F). As F_3 returns, the F_3 will get vanished from the recursive tree.

(d) Ans:

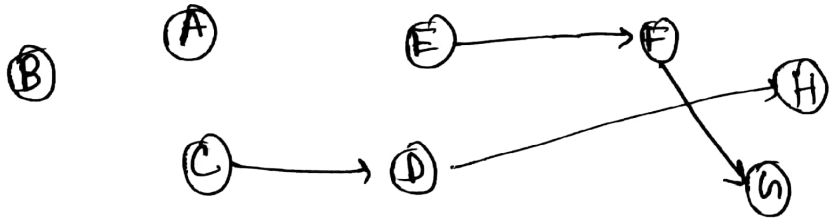
The graph,



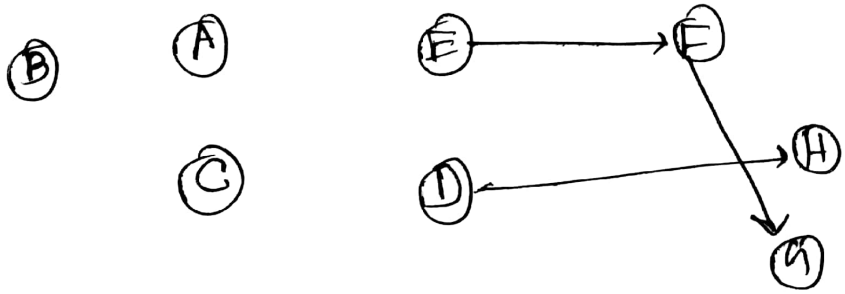
Applying Topological sort in Lexicographical order,



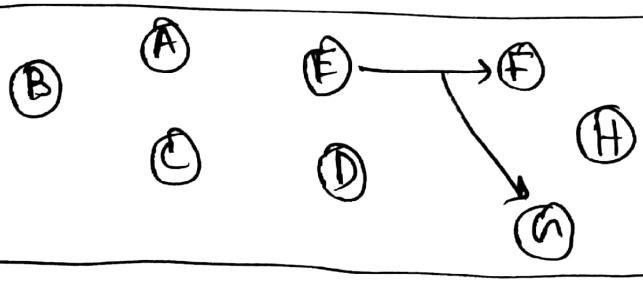
[Finished working B indegree 0]



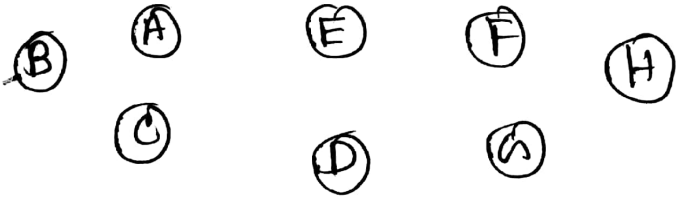
[A comes before C]



[C, as its indegree is 0]



[D]



∴ The order is : B → A → C → D → E → F → G → H