Analysis of Algorithm
(Dec 2019)

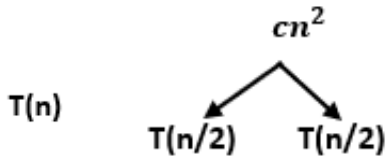
Q.P. Code - 78093

Q 1

a) Explain recurrences and various methods to solve recurrences. 5M

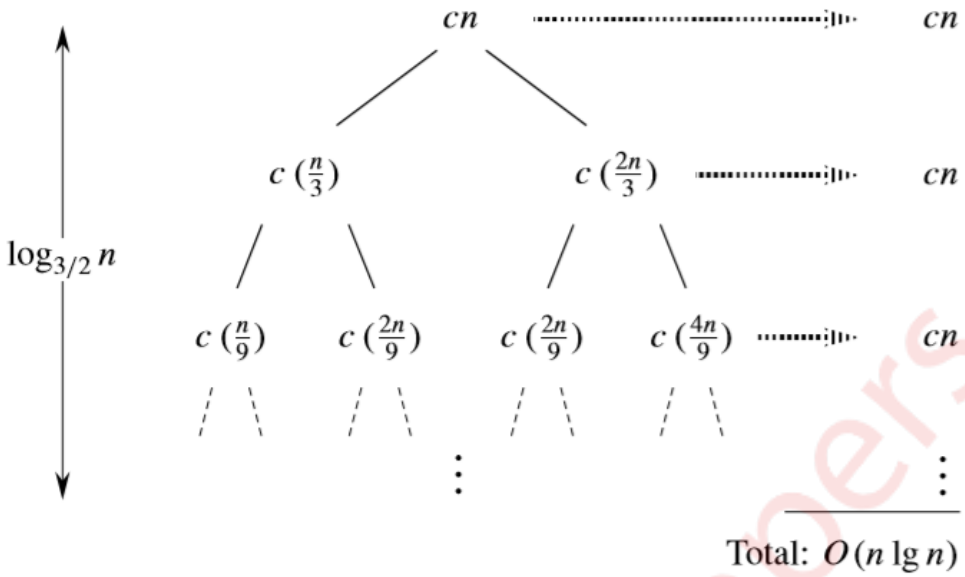
Definition:
Recurrence equation recursively defines a sequence of function with different argument, behavior of recursive algorithm is better represented using recurrence equations.
- Recurrence are normally of the form
T(n) = T(n-1) + f(n), for n > 1
T(n) = 0, for n = 0
- The function f(n) may represented constant or any polynomial in n.
- T(n) is interpreted as the time required to solve the problem of size n.
- Recurrence of linear search
T(n) = T(n-1) + 1
- Recurrence of selection/ bubble sort
- Recurrence is used in to represent the running time of recursive algorithms.
- Time complexity of certain recurrence can be easily solved using master methods.
- Substitution Method: Linear homogeneous recurrence of polynomial order greater than 2 hardly arises in practice.
- Two ways to solve unfolding method
i) Forward substitution ii) Backward substitution
- **Recursion Tree:**
- Recurrence tree method provides effective is difficult for complex recurrence.
- Ultimately, recurrence is the set of functions, each branch in recurrence tree represents the cost of solving one problem from the family of problems belonging to given recurrence.
Definition: Divide and conquer strategy uses recursion. The time complexity of the recursive program is described using recurrence. The master theorem is often suitable to find the time complexity of recursive problems.

- Master method is used to quickly solve the recurrence of the form T(n) = a . T(n/b) + f(n). Master method finds the solutions without substituting the values of T(n/b). In the above equation,
  n = Size of the problem
  a = Number of sub problems created in recursive solution.
  n/b = Size of each sub problem
  f(n) = work done outside recursive call.



a) T (n) **b) First level expansion of T(n)**

- A recursion tree for the recurrence T(n)= T(n/3)+T(2n/3)+cn.



b) Differentiate between P and NP **5M**

P:

- Polynomial time solving.
- Problems which can be solved in polynomial time, which take time like O(n), O(n2), O(n3).
- Example: finding maximum element in an array or to check whether a string is palindrome or not.
- So there are many problems which we can solved in polynomial time.
- P problems are set of problems that can be solved in polynomial time by deterministic algorithm.
- P is also known as PTIME and DTIME complexity class.
- Example of P problem: Insertion sort, Merge sort, Linear search, Matrix multiplication.
-

NP:

- Non deterministic Polynomial time solving. Problem which can't be solved in polynomial time like TSP (travelling salesman problem) or an easy example of this is subset sum: given a set of numbers, does there exist a subset whose sum is zero?
- But NP problems are checkable in polynomial time means that given a solution of a problem ,

NP-Complete:

- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.

- Now suppose we have a NP-Complete problem R and it is reducible to Q then Q is at least as hard as R and since R is an NP-hard problem. Therefore Q will also be at least NP-hard, it may be NP-complete also.
- NP complete is the combination of both NP and NP hard problem.
- Decision problem C is called NP complete if it has following two properties.
- C is in NP, and
- Every problem X in NP is reducible to C in polynomial time, i.e. For every X ε NP, $X \leq_p C$
  This two factor prove that NP-complete problems are the harder problems in class NP.
  They often referred as NPC.

c) Differentiate between Prims and Kruskals algorithm. 5M

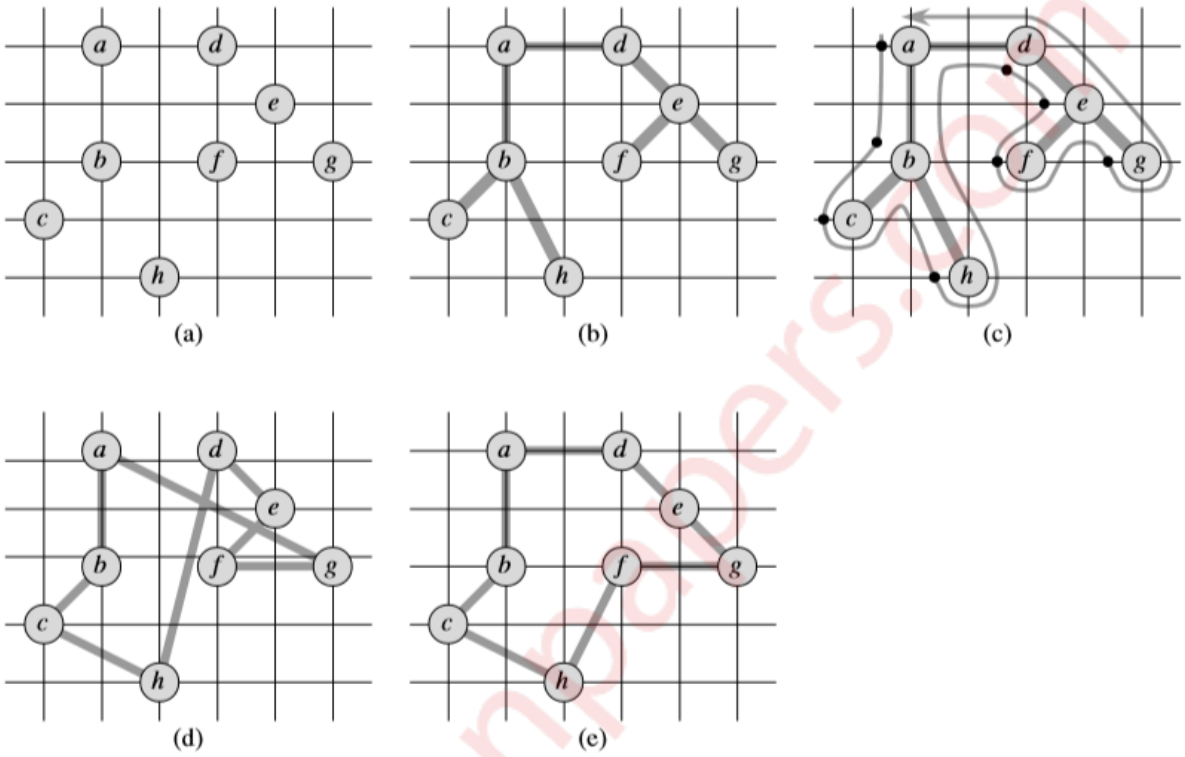
	Kruskal's Algorithm	**Prims Algorithm**
1.	This algorithm is for obtaining minimum spanning tree but it is not necessary to choose adjacent vertices of already selected vertices.	This algorithm is for obtaining minimum spanning tree by selecting the adjacent vertices of already selected vertices.
2.	Kruskal's algorithm initiates with an edge	Prim's algorithm initializes with a node
3.	Kruskal's algorithm select the edges in a way that the position of the edge is not based on the last step	Prim's algorithms span from one node to another
4.	Kruskal's can function on disconnected graphs too.	In prim's algorithm, graph must be a connected graph
5.	Kruskal's time complexity is O(logV).	Prim's algorithm has a time complexity of O(V2)
6.	Can run faster in sparse graph.	Can run faster in dense graph.
7.	Select the shortest edge,	Selects the root vertex.

d) Explain Dynamic Programming with example. 5M

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

Dynamic programming is basically, recursion plus using common sense. What it means is that recursion allows you to express the value of a function in terms of other values of that function. Where the common sense tells you that if you implement your function in a way that the recursive

calls are done in advance, and stored for easy access, it will make your program faster. It is memorizing the results of some specific states, which can then be later accessed to solve other sub-problems.
Example:



Q 2

a) Define Branch and Bound and explain 15 Puzzle problem. 10M

- Branch and bound builds the state space tree and find the optional solution quickly by pruning few of the tree branches which does not satisfy the bound.
- Backtracking can be useful where some other optimization techniques like greedy or dynamic programming fail.
- Such algorithms are typically slower than their counterparts. In the worst case, it may run in exponential time, but careful selection of bounds and branches makes an algorithm to run reasonably faster.
- In branch and bound, all the children of E nodes are generated before any other live node becomes E node.
- Branch and bound technique in which E node puts its children in the queue is called FIFO branch and bound approach.
- And if E node puts its children in the stack, then it is called LIFO branch and bound approach.
- Bounding functions are a heuristic function.
- Heuristic function computes the node which maximize the probability of better search or minimizes the probability of worst search.

- Used to solve optimization problems.
- Nodes in tree may be explored in depth-first or breadth-first order.
- Next move is always towards better solution.
- Entire state space tree is searched in order to find optimal solution.
- In this problem there are 15 tiles, which are numbered from 0 – 15.

- initial and goal arrangement

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

a) Initial state

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

b) Goal state

- There is always an empty slot in the initial state.
- In the initial state moves are the moves in which the tiles adjacent to ES are moved to either left, right, up or down.
- Each move left, right, up and down creates a **new** arrangement in a tile.
- These arrangements are called different states of the puzzle.
- The initial arrangement is called as initial state and goal arrangement is called as goal state.
- The state space tree for 15 puzzle is very large because there can be 16! Different arrangements.
- In state space tree, the nodes are numbered as per the level.
- Each next move is generated based on empty slot positions.
- Edges are label according to the direction in which the empty space moves.
- The root node becomes the E – node.
- The child node 2, 3, 4 and 5 of this E – node get generated.
- Out of which node 4 becomes an E – node. For this node the live nodes 10, 11, 12 gets generated.
- Then the node 10 becomes the E – node for which the child nodes 22 and 23 gets generated.
- We can decide which node become an E – node based on estimation formula.
- Cost estimation formula
  C (x) = f (x) + G (x)

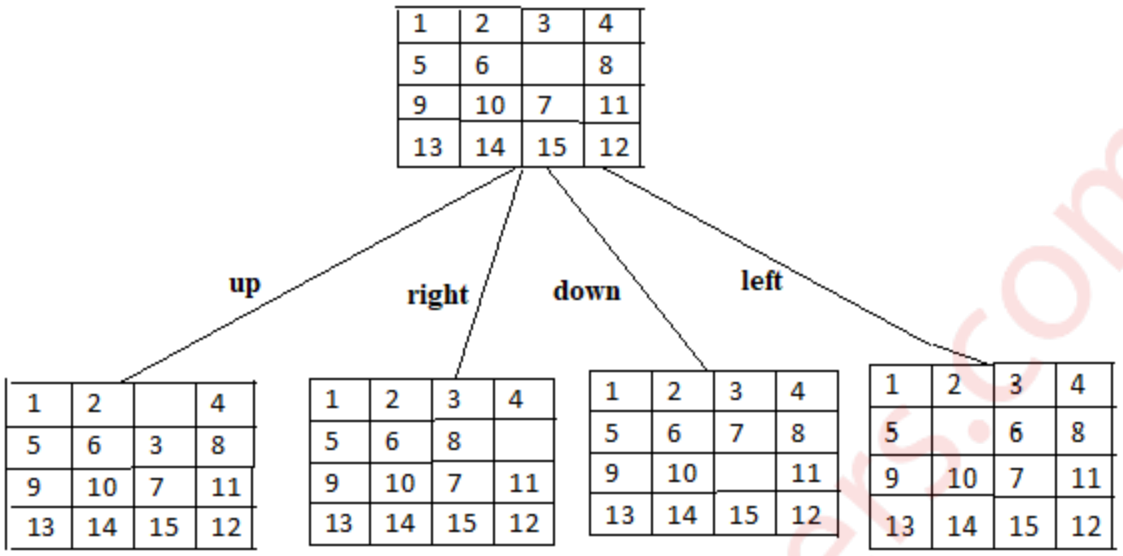


Fig. Cost f(x) + h(x) after first move

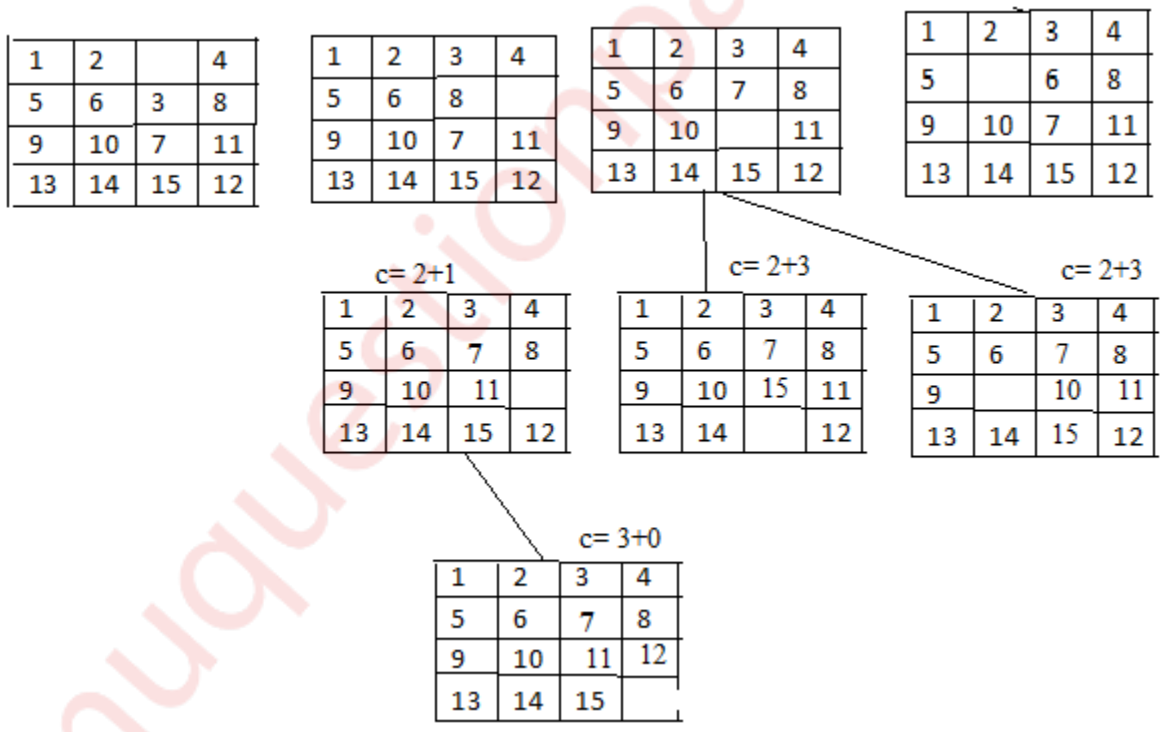
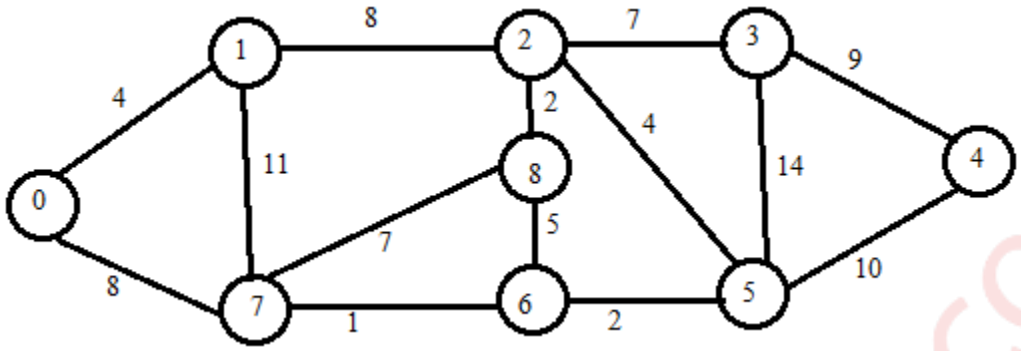


Fig. Tile positions after subsequent moves

b) Apply Dijkstra's algorithm on the following graph. Consider vertex 0 as sources. **10M**



Visited Vertices	dist[u]								comment
	1	2	3	4	5	6	7	8	
0	4	∞	∞	∞	∞	∞	8	∞	Select the vertex having minimum distance i.e 1
0 -1	4	Min (∞, 4 + 8)=12	∞	∞	∞	∞	Min(8, 4+ 11)= 8	∞	Visit remaining vertices from 0 via 1 and update distance and go to vertex having minimum distance from 1 i.e 7
0-1-7	4	12	∞	∞	∞	Min(∞, 8+1)=9	8	Min(∞, 8+7)=15	Visit remaining vertex via 0 1 and 7 and update distance and go to vertex having minimum from 7 distance i.e 6
0-1-7-6	4	12	∞	∞	Min(∞, 9+2)=10	9	8	Min(15, 9+5)=14	Visit remaining vertex via 0,1,7,6 and update distance and go to vertex having minimum distance from 6 i.e 5
0-1-7-6-5	4	Min (12, 10+4)=12	Min(∞, 10+14)=24	Min(∞, 10+10)=20	10	9	8	14	Visit remaining vertex via 0,1,7,6 ,5and update distance and go to vertex having

									minimum distance from 5 i.e 2
0-1-7-6-5-2	4	12	Min(24, 12+7)=19	20	Min(10, 12+4)=16	9	8	14	Visit remaining vertex via 0,1,7,6 ,5,2and update distance and go to vertex having minimum distance from 2 i.e 5

Shortest path for the given graph is 0-1-7-6-5-2

Q 3

a) Find longest common subsequence for following strings: 10M

X= ababcde

Y= bacadb

i) The longest common sequence is the problem of finding maximum length common subsequence from given two string A and B.

ii) Let A and B be the two string. Then B is a subsequence of A. a string of length m has 2^m subsequence.

iii) This is also one type of string-matching technique. It works on brute force approach.

iv) Time complexity = O(m*n)

Algorithm **LONGEST_COMMON_SUBSEQUENCE (X, Y)**

```
// X is string of length n and Y is string of length m
for i ← 1 to m do
   LCS [i, 0] ← 0
   end
for j ← 0 to n do
    LCS [0, j] ← 0
   end
for i ← 1 to m do
   for j ← 1 to n do
     if Xi = = Yj then
        LCS [i, j] = LCS [i-1, j-1] +
     else if LCS [i-1, j] ≥ LCS [i, j-1]
        LCS [i, j] = LCS [i-1, j]
     else
         LCS [i, j] = LCS [i, j-1]
    end
   end
  end
end
return LCS
```

P →

Q ↓

			1	2	3	4	5	6	7
			a	b	a	b	c	d	e
		0	0	0	0	0	0	0	0
1	b	0	0	1	1	2	2	2	2
2	a	0	1	1	2	2	2	2	2
3	c	0	1	1	2	2	3	3	3
4	a	0	2	2	3	3	3	3	3
5	d	0	2	2	3	3	3	4	4
6	b	0	2	3	3	4	4	4	4

So the **LCS is (b, a, c, d)**

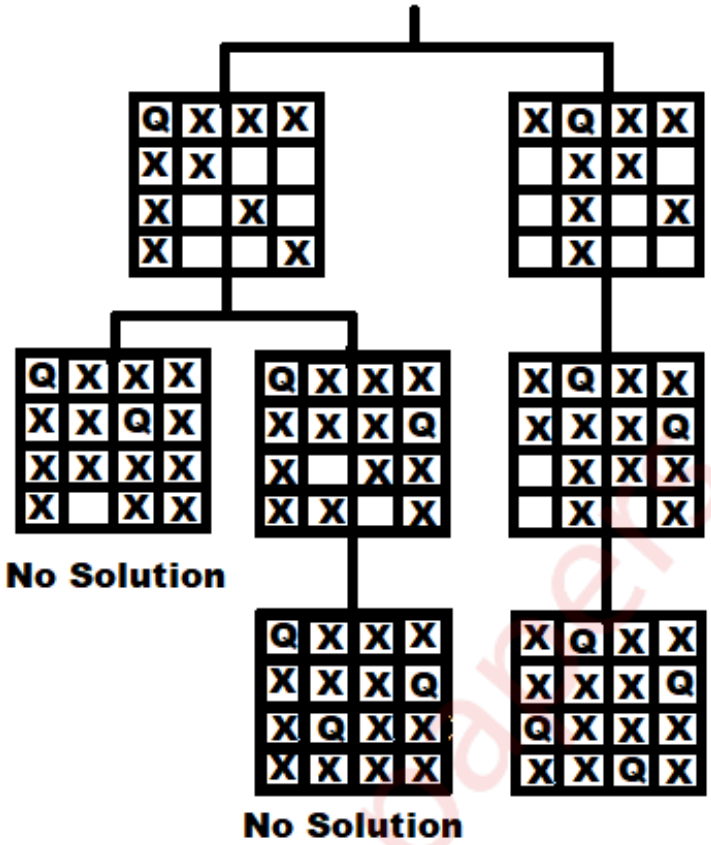
b) Explain Backtracking with N-queen problem **10M**

i) n-queens problem is a problem in which n-queens are placed in n*n chess board, such that no 2 queen should attack each other.

ii) 2 queens are attacking each other if they are in same row, column or diagonal.

iii) For simplicity, State space tree is shown below. Queen 1 is placed in a 1st column in the 1st row. All the position is closed in which queen 1 is attacking. In next level, queen 2 is placed in 3rd Column in row 2 and all cell are crossed which are attacked by already placed 1 and 2. As can be seen below. No place is left to place neat queen in row 3, so queen 2 backtracks to next possible and process continue.

iv) In a similar way, if (1, 1) position is not feasible for queen 1, then algorithm backtracks and put the first queen cell (1, 2), and repeats the procedure. For simplicity, only a few nodes are shown below in state space tree.



v) Complete state space tree for the 4 – queen problem is shown above. 2 – queen problem is not feasible.

vi) This is how backtracking is used to solve n-queens problems.

Q 4

a) Formulate Knapsack problem, Explain and differentiate between greedy knapsack and 0/1 knapsack. **10M**

- Given a set of items, each having different weight and value or profit associated with it. Find the set of items such that the total weight is less than or equal to capacity of the knapsack and the total value earned is as large as possible.

-The knapsack problem is useful in solving resource allocation problem.

-Let X=<$x_1, x_2, x_3, \ldots\ldots, x_n$>be the set of items. Sets W=<$w_1, w_2, w_3, \ldots\ldots, w_n$>and V=<$v_1, v_2, v_3, \ldots\ldots, v_n$> are weight and value associated with each items in X. Knapsack capacity is M unit. The knapsack problem is to find the set of items which maximizes the profit such that collective weight of selected items does not cross the knapsack capacity.

-Select items from X and fill the knapsack such that it would maximize the profit. Knapsack problem has two variations. 0/1 knapsack, that does not allow breaking of items . Either add an entire item or reject it. It is also known as a binary knapsack. Fractional knapsack allows breaking of items. Profit will be earned proportionally.

Mathematical formulation

We can select any items only ones. We can put items Xi in knapsack if knapsack can accommodate it. If the item is added to a knapsack, associated profit is accumulated.

Objective is to maximize $\sum_{i=1}^{n} x_i\, v_i$, subjected to $\sum_{i=1}^{n} x_i w_{i \le M}$

Where,

v_i = Profit associated with i^{th} item

w_i = Weight of i^{th} items

M=Capacity of knapsack

x_i=Fraction of i^{th} item

={0,1}→ 0/1 knapsack

=[0,1]→fractional knapsack.

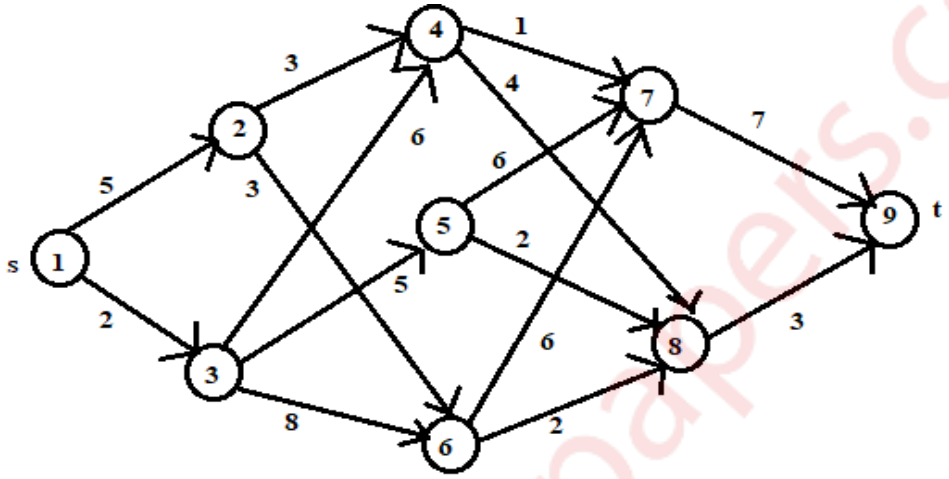
We will discuss two approaches for solving knapsack using dynamic programming.

0/1 knapsack	**Fractional knapsack**
1. It is a part of dynamic programming	1. It is a part of greedy algorithm.
2. Dynamic Programming is used to obtain the optimal solution.	2. Greedy Method is also used to get the optimal solution.
3. In dynamic programming we choose at each step but the choice may depend the solution to sub-problems.	3. In a greedy algorithm we make whatever choice seen best at the movement and then solve the sub problem arising then the choice is made.
4. Less efficient as compared to a greedy approach	4. More efficient as compared to a greedy approach
5. It is guaranteed that Dynamic Programming will generate an optimal solution using Principle of Optimality.	5. In Greedy Method, there is no such guarantee of getting Optimal Solution.

b) Explain Multistage graph with example. 10M

- A **Multistage graph** is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only In other words there is no edge between vertices of same stage and from a vertex of current stage to previous stage.
- The **Brute force** method of finding all possible paths between Source and Destination and then finding the minimum. That's the WORST possible strategy

- **Dijkstra's Algorithm** of Single Source shortest paths. This method will find shortest paths from source to all other nodes which is not required in this case. So it will take a lot of time and it doesn't even use the SPECIAL feature that this MULTI-STAGE graph has.
- **Simple Greedy Method** – At each node, choose the shortest outgoing path. If we apply this approach to the example graph give above we get the solution as 1 + 4 + 18 = 23. But a quick look at the graph will show much shorter paths available than 23. So the greedy method fails.
- The best option is Dynamic Programming. So we need to find **Optimal Sub-structure, Recursive Equations and Overlapping Sub-problems.**
- Example of multistage graph:



Stage 1:
Vertex 1 is connected to 2 and 3
Cost[1] = min{c[1, 2] , c[1, 3] }
= min {5, 2}
= 2

Stage 2:
Vertex 2 is connected to 4 and 6
Cost[2] = min {c[2,4], c[2, 6]}
= min{3, 3}
= 3
Vertex 3 is connected to 4, 5 and 6
Cost [3]= min{c[3,4], c[3,5], c[3, 6]}
= min{ 6, 5, 8}
= 5

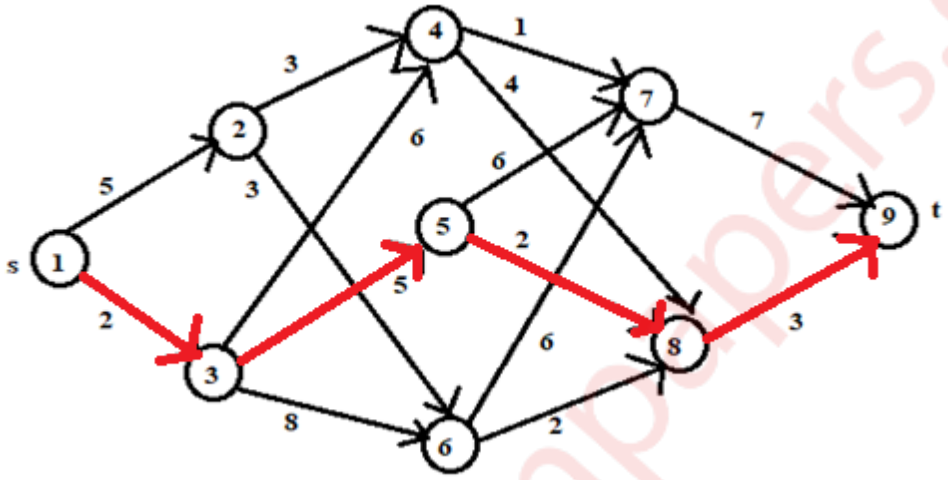
Stage 3:
Vertex 4 is connected 7 and 8
Cost [4] = min{c[4,7], c[4,8]}
=min {1, 4}
= 1
Vertex 5 is connected to 7 and 8
Cost [5]= min{c[5,7], c[5, 8]}
= min{6,2}
= 2

Vertex 6 is connected to 8
Cost [6]= min{c[6, 8]}
= 2

Stage 4:

Vertex 7 is connected to 9
Cost [7]= {c[7, 9]}
= 7
Cost [8]= {c[8, 9]}
= 3

Minimum cost path from s to t



Q 5

a) Rewrite KMP algorithm and explain with example. 10M

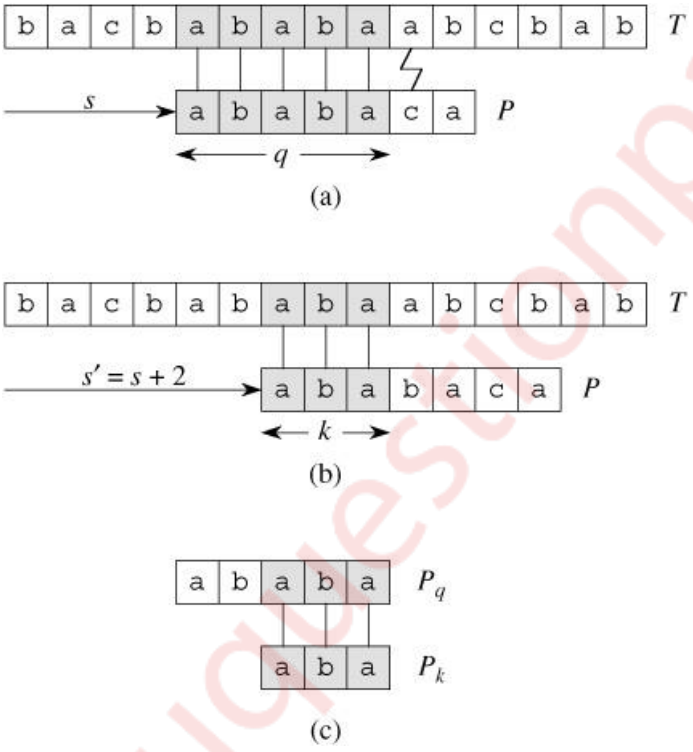
a. This is first linear time algorithm for string matching. It utilizes the concept of naïve approach in some different way. This approach keeps track of matched part of pattern.
b. Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive approach.
c. This algorithm builds a prefix array. This array is also called as ∏ array.
d. Prefix array is build using prefix and suffix information of pattern.
e. This algorithm achieves the efficiency of O(m+n) which is optimal in worst case.

Algorithm – **KNUTH_MORRIS_PRATT (T, P)**

```
n = T.length
m = P.length
∏ = Compute prefix
q ⍰ 0
for i = 1 to n
while q > 0 and P[q+1] ≠ T[i]
q = ∏ [q]
```

if P[q+1] = = T[i]
p= q+1
if q = = m
Print "pattern found"
q = ∏ [q]
COMPUTE_PREFIX (P)
M = P.length
Let ∏ [1……m] be a new array
∏ [1] = 0
K = 0
for k = 0 to m
while k > 0 and P[k+1] ≠ T[q]
k = ∏ [k]
if P[k+1] = = T[q]
k = k + 1
∏ [q] = k
return ∏

Example:



b) Define chromatic number of graphs, Explain graph coloring algorithm. 10M

-Defination of chromatic number:

The Smallet Number of color required for coloring graph is called its chromatic number.

-The chromatic number is denoted by X(G). Finding the chromatic number for the graph is NP-complete problem.

-Graph coloring problem is bith, decision problem as well as an optimization problem. A decision problem is stated as, “With given M colors and graph G, whether such color scheme is possible or not?”.

-The optimization problem is stated as, “Given M colors and graph G, find the minimum number of colors required for graph coloring.”

-Graph coloring problem is a very interesting problem of graph theory and it has many diverse applications.

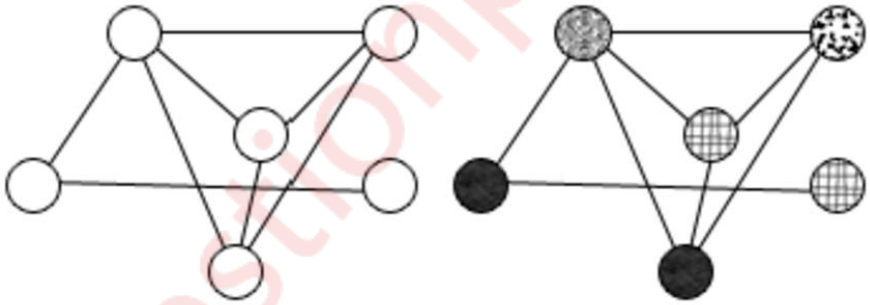
-The problem can be solved simply by assigning a unique color to each vertex,but this solution is not optimal. It may be possible to color the graph with colors less than IvI.

-This problem can be solved using backtracking algorithms as follows:

- List down all the vertices and colors in two lists
- Assign color 1 to vertex1
- If vertex 2 is not adjacent to vertex 1 to then assign the same color, otherwise assign color2.
- Repeat the process until all vertices are colored.

-Algorithm backtracks whenever color i is not possible to assign any vertex k and it selects next color i+1 and test is repeated.

- Example:



Q 6

a) Master Theorem **5M**

Definition: Divide and conquer strategy uses recursion. The time complexity of the recursive program is described using recurrence. The master theorem is often suitable to find the time complexity of recursive problems.

- Master method is used to quickly solve the recurrence of the form T(n) = a . T(n/b) + f(n).

Master method finds the solutions without substituting the values of T(n/b). In the above equation,
n = Size of the problem
a = Number of sub problems created in recursive solution.
n/b = Size of each sub problem
f(n) = work done outside recursive call.

This includes cost of division of problem and merging of the solution.

Example:

Given: $T(n) = 8T(n/2) + n^2$

Compare this equation with $T(n) = a\,T(n/b) + f(n)$

Here, $a = 8$, $b = 2$ and $f(n) = n^2$

Checking the relation between a and b^2

As $a > b^2$

i.e $8 > 2^2$, **Solution for this equation is given as,**

T(n) = $\left(n^{log_b^a}\right)$ **=** $\theta\left(n^{log_2^8}\right)$ **=** $\theta\left(n^{log_2^{2^3}}\right)$ **=** $\theta\left(n3^{log_2^2}\right)$ **=** $\theta\,(n^3)$

b) Rabin Karp algorithm **5M**

i) It is based on hashing technique.
ii) It first compute the hash value of pattern and text. If hash values are same,i.e if hash(p) = hash(t). we check each character if characters are same pattern is found. If hash value are not same no need of comparing string.
iii) Strings are compared using brute force approach. If pattern is found then it is called as Hit. Otherwise it is called as Spurious Hit.Time Complexity = O(n), for worst case sometimes it is O(mn) when prime number is used very small.

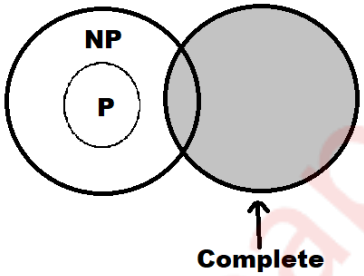
Algorithm – **RABIN_KARP (T, P)**

```
n = T.length
m = P.length
hᵖ = hash(T
hᵗ = hash(T) (0……m-1)
for S=0 to n-m
 if (hᵖ = hᵗ)
 if (P(0…..m-1) == T(0…..m-1))
 print "Pattern Found"
 if (S < n-m)
  hᵗ = hash(S+1……S+m-1)
```

c) Steps for NP Completeness proofs **5M**

NP-Complete:

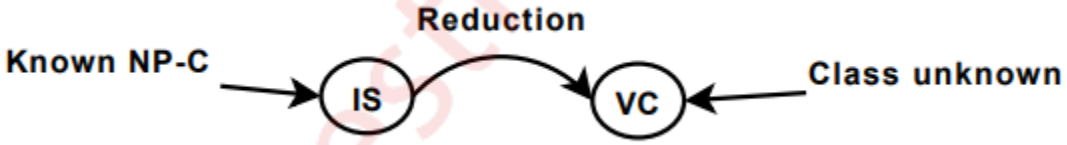
- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.
- Now suppose we have a NP-Complete problem R and it is reducible to Q then Q is at least as hard as R and since R is an NP-hard problem. therefore Q will also be at least NP-hard , it may be NP-complete also.
- NP complete is the combination of both NP and NP hard problem.
- Decision problem C is called NP complete if it has following two properties.
- C is in NP, and
- Every problem X in NP is reducible to C in polynomial time, i.e. For every X ε NP, $X \leq_p C$
  This two factor prove that NP-complete problems are the harder problems in class NP.
  They often referred as NPC.



NP Complete

Select a known N P-complete problem.

- Independent Set (IS) is a known N P-complete problem.
- Use IS to prove that VC is N P-complete.



- Define a polynomial-time reduction from IS to VC:

- Given a general instance of IS: G 0=(V 0 , E 0), k 0
- Construct a specific instance of VC: G=(V, E), k I V =V 0 I E=E 0 I (G=G 0) I k=|V 0 | − k 0
- This transformation is polynomial:
- Constant time to construct G=(V, E)
- O(|V|) time to count the number of vertices
- Prove that there is a VI (|VI | = k 0) for G 0 iff there is an VC (|VC | = k) for G.

d) Assembly Line Scheduling **5M**

- Assembly line scheduling is a manufacturing problem. In automobile industries assembly lines are used to transfer parts from one station to another station.

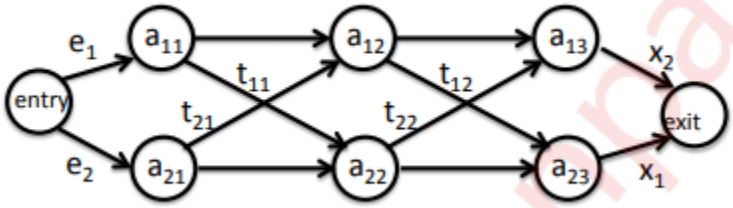
- Manufacturing of large items like car, trucks etc. generally undergoes through multiple stations, where each station is responsible for assembling particular part only. Entire product be ready after it goes through predefined n stations in sequence.

- Manufacturing of car may be done through several stages like engine fitting, coloring, light fitting , fixing of controlling system, gates, seats and many other things.

- The particular task is carried out at the station dedicated to that task only. Based on the requirement there may be more than one assembly line.

- In case of two assembly lines if the load at station j at assembly 1 is very high, then components are transfer to station j of assembly line 2 the converse is also true. This technique helps to speed ups the manufacturing process.

- The time to transfer partial product from one station to next station on the same assembly line is negligible. During rush factory may transfer partially completed auto from one assembly line to another, complete the manufacturing as quickly as possible.



e) Strassen's matrix multiplication **5M**

- Strassen has proposed divide and conquer strategy-based algorithm, which takes less numbers of multiplications compare to this traditional way of matrix multiplication.
- Using Strassen's method, multiplication operation is defined as,

$\begin{bmatrix} C11 & C12 \\ C21 & C22 \end{bmatrix} = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} * \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix}$

C11 = S1+ S4 – S5 + S7
C12 = S3+ S5
C21 = S2 + S4
C22 = S1 + S3 – S2 +S6
Where,
S1 = (A11 + A22) * (B11 + B22)
S2 =(A21 + A22) *B11
S3 = A11 *(B12 - B22)
S4 = A22 *(B21 – B11)
S5 = (A11 + A12) * B22
S6 = (A21 -A11) * (B11 +B12)
S7 = (A12 -A22) * (B21 +B22)
Let us check if it same as conventional approach.

C12 = S3 + S5

= A11 * (B12 – B22) + (A11 + A12) * B22

= A11 * B12 + A12 * B22

This is same as C12 derived using conventional approach. Similarly we can derive all Cij for Strassen's matrix multiplication. Algorithm for Strassen's multiplication.

Complexity:

Conventional approach performs eight multiplications to multiply two matrices of size 2*2. Whereas Strassen's approach performs seven multiplications on problem of size 1 * 1, which in turn finds the multiplication of 2 * 2 matrices using addition.

Strassen's approach creates seven problems of size (n-2).

Recurrence equation for Strassen's approach is given as,

T(n) = 7T ($\frac{n}{2}$)

Two matrices of size 1 * 1 needs only one multiplication, so best case would be, T(1) =1.

Let us find solution using iterative approach. By substituting n =(n/2) in above equation,

$$T(\frac{n}{2}) = 7T(\frac{n}{4})$$

$$T(n) = 7^2 T\left(\frac{n}{2^2}\right)$$

.

.

$$T(n) = 7^k T\left(\frac{n}{2^k}\right)$$

Let's assume n = 2^k

$$T(2^k) = 7^k T\left(\frac{2^k}{2^k}\right) = 7^k . T(1) = 7^k = 7^{log_2 n}$$

$$= n^{log_{2^7}} = n^{2.81} < n^3$$

Thus, running time of Strassen's matrix multiplication algorithm **O($n^{2.81}$)**

****