**Analysis of Algorithm Previous Year Question Paper**

**Solved + Unsolved from May2018 to May 22**

# Analysis of Algorithm
# (May 2018)

**Q.P. Code - 38841**

**Q 1 Answer the following**

**a. Write the difference between greedy method and dynamic programming. 5M**

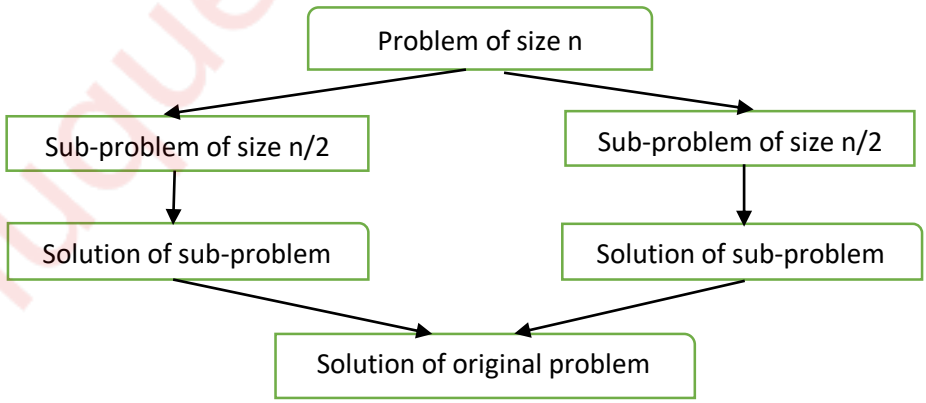
| Greedy method | Dynamic programming |
|---|---|
| 1. Greedy method does not guarantee an optimal solution. | 1. Dynamic programming guarantees an optimal solution. |
| 2. Sub-problems do not overlap. | 2. Sub-problems overlap. |
| 3. It does little work. | 3. It does more work. |
| 4. Only considers the current choices. | 4. Considers the future choices. |
| 5. Construct the solution from the set of feasible solutions. | 5. There is no specialized set of feasible choices. |
| 6. Select choice which is locally optimum. | 6. Select choices which is globally optimum. |
| 7. There is no concept of memorization. | 7. Employ memorization. |
| 8. Examples- Knapsack problems, Job scheduling with deadlines, Kruskal , Prim's. | 8. Examples- All pair shortest path, 0/1 Knapsack, Travelling salesman problem, LCS. |

**b. Explain the general procedure of divide and conquer method. 5M**

Divide and Conquer method: This is the most widely applicable technique for designing efficient algorithms. It works in three stages as shown below.

1) **Divide**: Recursively divide the bigger problem of size n into smaller sub-problems of size n/2
2) **Solve**: Sub-problems are solved independently.
3) **Combine**: Combine solutions of smaller sub-problems to derive the solution of larger big problem of size n.

Smaller sub-problems are similar to the larger problem with smaller arguments. Hence such Problems can be solved easily using recursion. Divide and conquer is multi-branched, Top-Down recursive approach. Each branch indicates one sub-problems and it calls itself with the Smaller argument.




Above diagram shows the working of Divide and Conquer method. Sub-problem may or may not be of size n/2.

**c. Determine the frequency counts for all statement in the following algorithm Segment.** **5M**

```
I=1;
While(I<=n)
{
X=X+I;
I=I+1;
}
```

| | S/e | Frequency | Total Steps |
|---|---|---|---|
| I=1; | 1 | 1 | 1 |
| While(I<=n) | 1 | I + n | I + n |
| { | 0 | - | - |
| X=X+I; | 1 | I | I |
| I=I+1; | 1 | 1 | 1 |
| } | 0 | - | - |
| | | | **2(I+1) + n.** |

**d. What is backtracking Approach? Explain how it is used in Graph Coloring.** **5M**

**Backtracking Approach:**

1) Backtracking is the process where the entire problem is divided into several stages. The algorithm then attempts to find the solution to the problem by constructing partial solutions remain consistent with the requirements of the problem.
2) However, when an inconsistency with the requirement of the problem occurs the algorithm backs up (backtracks) by removing the most recently constructed part of the solution and trying another possibility.
3) For example, the algorithm will first find solution for stage one and stage two and so on till stage N. However, if it cannot find the solution for stage N+1, then it will go back to stage N, remove the solution it found for stage N and will come up with a new solution for stage N.
4) Examples: Graph Coloring & n-Queens problem.

**Graph Coloring:**

1) Coloring the vertices of the graph in such a way such that no two adjacent vertices have the same color. This is called Graph Coloring.
2) It is also called as Vertex coloring or K-coloring.
3) The smallest number of color required for coloring the graph is called as Chromatic number.
4) Example:

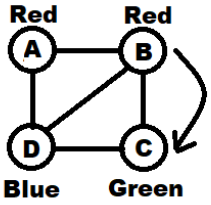


Fig. A

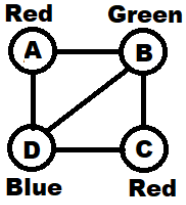


Fig. B

In Fig. A same color(Red) is assigned to adjacent vertices (A & B) which is against rule so we Backtrack and change the color as shown in the Fig. B. We cannot assign another color on the

Place of Red because chromatic color should be minimum. If we assign another color on the Place of Red Chromatic Number Will be 4 for Fig. A. and if we see Fig B. chromatic number is 3 Which is minimum. Fig. B is Right solution for Graph coloring,

Time Complexity – $O(m^n)$

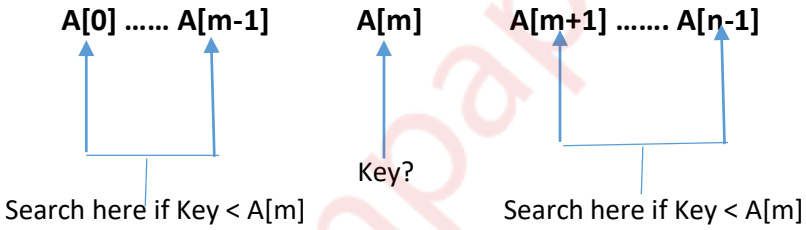
---

**Q 2.a. Explain with example how divide and conquer strategy is used in binary Search? 10M**

Binary search is an efficient searching method. While searching the elements using this method the most essential thing that the elements in the array should be sorted one. An element which Is to be sorted from the list of elements sorted in array should be sorted one. An element which is to be searched from the list of elements stored in array A [0…n-1] is called KEY element. Let A [m] be the mid element of array A. then these are three conditions that needs to be satisfied while searching the array using this method.

1) If KEY=A[m] then desired element is present in the list.
2) Otherwise if KEY<A[m] then search the left sub list
3) Otherwise if KEY>A[m] then search the right sub list.

This can be represented as




As shown above the array is divided into two part left and right sub list according to key element We decide in which list we can find the element. This is how divide and conquer strategy is used in Binary search.

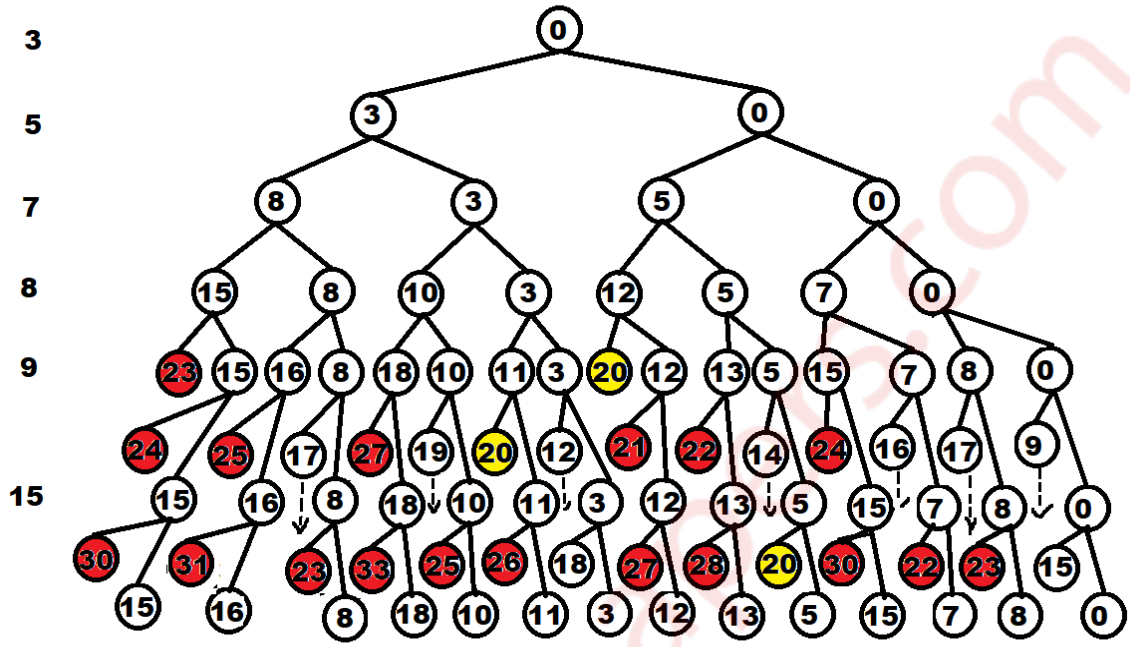
Algorithm: **BINARY_SEARCH(A, Key)**

```
Low <- 1
High <- n
While low < high do
    Mid <- (low + High) / 2
  If A[Mid] == Key then
     return Mid
  else if A[Mid] < Key then
     Low <- mid + 1
  else
     High <- Mid – 1
    end
  end
return 0
```

Time Complexity – Best case =O(1), Average case & worst case = $O(\log_2 n)$

**b. Solve sum of subsets problem for following** **10M**

**N=6 W={3,5,7,8,9,15} & M=20 Also write the Algorithm for it.**




**Solutions** : Subset **1) {5, 7, 8}**

**2) {9, 8, 3}**

**3) {5, 15}**

**Algorithm:**

Let W be a set of elements & M be the expected sum of subset then

**Step1**: Start with empty set.

**Step2**: Add to the subset, the next element from the list.

**Step3**: If the subset is having sum equal to M then Stop with that subset as solution,

**Step4**: If the subset is not matching with the M or if we have reached to the end of the set Then backtrack through that subset until we find suitable value.

**Step5**: If the subset is less then M then repeat Step2.

**Step6**: If we have visited all the elements without finding suitable & No backtrack is possible Then stop without solution.

Time Complexity – $O(2^n)$

---

**Q.3.a. Obtain the solution to knapsack problem by Greedy method n=7, m=15 (p1, P2,…P7)=(10,5,15,7,6,18,3), (W1,W2,…W7)=(2,3,5,7,1,4,1)** **10M**

| Item | Weight | Value | Value / Weight |
|---|---|---|---|
| P1 | 2 | 10 | 5 |
| P2 | 3 | 5 | 1.667 |
| P3 | 5 | 15 | 3 |
| P4 | 7 | 7 | 1 |
| P5 | 1 | 6 | 6 |
| P6 | 4 | 18 | 4.5 |
| P7 | 1 | 3 | 3 |

Arrange the Item in increasing order of value / Weight Ratio
P5, P1, P6, P7, P3, P2, P4.
Capacity of Bag is 15.

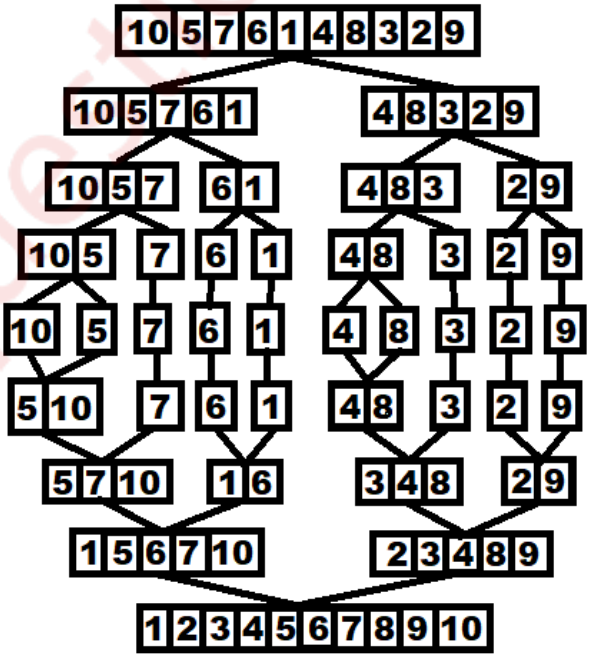
| Capacity of Bag | Items | value |
|---|---|---|
| 15 | ----- | 0 |
| 14 | P5 | 6 |
| 12 | P5, P1 | 16 |
| 8 | P5, P1, P6 | 34 |
| 7 | P5, P1, P6, P7 | 37 |
| 2 | P5, P1, P6, P7, P3 | 52 |

Now, P2 arrives but capacity of bag is only 2 and weight is 3 so we are going to take fraction

**Maximum Profit** = $52 + \frac{2}{3} * 5 =$ **55.33**

**b. Sort the list of the elements 10,5,7,6,1,4,8,3,2,9 using merge sort algorithm and show its computing time is O(n logn).** **10M**

Merge sort 10, 5, 7, 6, 1, 4, 8, 3, 2, 9



**Complexity Analysis**

Using Master method: $T(n) = 2\,T\left(\frac{n}{2}\right) + n$

Comparing with : $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

a = 2, b = 2 & f(n) = n with k = 1

where k is degree pf polynomial function f(n)

$a = b^k$, $2 = 2^1$.

So from the case I variant 2

$T(n) = O(n^{\log_2 2} \log n)$

**T(n) = O(n log n)**

"Hence Proved"

Time Complexity - $T(n) = O(n \log_2 n)$

---

**Q.4.a. Explain different string matching algorithms.** **10M**

There are various String matching algorithms listed below.

**A] Naive:**

i) It is the simplest method which uses brute force approach.

ii) It is a straight forward approach of solving the problem.

iii) It compares first character of pattern with searchable text. If match is found, pointers in both strings are advanced. If match not found, pointer of text is incremented and pointer of pattern is reset. This process is repeated until the end of the text.

iv) It does not require any pre-processing. It directly starts comparing both strings character by character.

v) Time Complexity = O(m*(n-m))

Algorithm -- **NAVE_STRING_MATCHING (T, P)**

```
for i ← 0 to n-m do
    if P[1…….m] = = T[i+1…..i+m] then
        print "Match Found"
    end
end
```

**B] Rabin-Karp:**

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.

iv) 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
h^p = hash(T
h^t = hash(T) (0……m-1)
for S=0 to n-m
 if (h^p = h^t)
if (P(0…..m-1) == T(0…..m-1))
 print "Pattern Found"
if (S < n-m)
 h^t = hash(S+1……S+m-1)
```

**C) Finite Automata:**

i) Idea of this approach is to build finite automata to scan text T for finding all occurrences of of pattern P.

ii) This approach examines each character of text exactly once to find the pattern. Thus it takes linear time for matching but preprocessing time may be large.

iii) It is defined by tuple M = {Q, ∑, $q_o$, F, ∂}

Where Q = Set of States in finite automata

∑ = Sets of input symbols

$q_o$ = Initial state

F = Final State

∂ = Transition function

iv) Time Complexity = O($M^3$|∑|)

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

```
State ← 0
for I ← 1 to n
State ← ∂(State, t_i)
   If State == m then
      Match Found
   end
end
```

**D) Knuth Morris Pratt (KMP)**

i) 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.

ii) Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive approach.

iii) This algorithm builds a prefix array. This array is also called as ∏ array.

iv) Prefix array is build using prefix and suffix information of pattern.

v) 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]
     q = q+1
  if q = = m
Print "pattern found"
   q = ∏ [q]
```

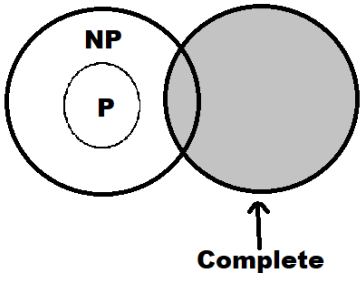
```
COMPUTE_PREFIX (P)
M = P.length
Let ∏ [1……m] be a new array
 ∏ [1] = 0
 K = 0
for k = 0 to m
   while k > 0 and P[k+1] ≠ T[q]
     k = ∏ [k]
    if P[k+1] = = T[q]
     k = k + 1
   ∏ [q] = k
    return ∏
```

**b. What do you understand by NP Complete? Explain is subset sum problem NP complete? If so explain.** **10M**

i) NP complete is the combination of both NP and NP hard problem.

ii) 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

Sum of Subset:

i) Sum of subset is NP complete.

ii) It satisfy the above two condition. The problem cannot be solved in polynomial time but can be verified in polynomial time hence it is NP. Every problem in NP can be reduced in polynomial time hence it is NP hard. Therefore, sum of subset id NP complete.

Example: set- { 3, 5, 7, 8, 9, 15}

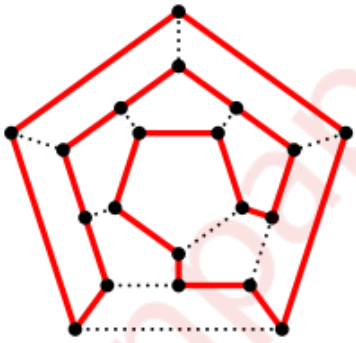
Time complexity = $O(2^n) = 2^6 = 64$ which is quiet large.

So it requires more time to compute then polynomial time. But its verification is easy it Can be done into polynomial time.

---

**Q.5.a. write a detailed note on Hamiltonian cycles.** **10M**

- The Hamiltonian cycle of undirected graph G = <V, E> is the cycle containing each vertex in V.
- If graph contains a Hamiltonian cycle, it is called Hamiltonian graph otherwise it is non-Hamiltonian.



The dodecahedron shows the Hamiltonian, and the Hamiltonian cycle is shown by thick red lines. Whereas the bipartite graph with an odd number of vertices as shown above.

- One way of checking if the graph is Hamiltonian or not is to list out all possible permutation of Vertices and check them one by one. There are m! different permutation of m vertices, and hence The running tome of algorithm would be $\Omega(m!)$ time. By encoding the graph using its adjacency Matrix representation, we can reduce the time $\Omega(\sqrt{N}) = \Omega(2^{\sqrt{N}})$ which is not polynomial.
- Here n represents the length of encoding of graph G. Thus, the given problem cannot be solved in Polynomial time.

**Verification**

- If given the solution string of Hamiltonian graph, it is very easy to prove the correctness of it. We Just need to check if the solution contains all the vertices of V and there must be an edge Between two consecutive vertices. This can be done in $O(n^2)$ time. Thus, the solution can be Verified in polynomial time., where n is the length of graph G. Verification algorithm A(x, y) is defined by two arguments, where x is the input string and y is the Binary string, called certificate y such that A(x, y) = 1, we say that algorithm verifies the string x. And the language defined by the algorithm is, $L = \{x \in \{0, 1\}^* : \text{there exist } y \in \{0, 1\} \text{ such that } A(x, y) = 1\}$
- For each legal string $x \in L$, A must verify the string and produce the certificate y. And for any string x which is not in L, must not verify the string, and no certificate can prove that $x \in L$. For example if the graph is Hamiltonian, the proof can be checked in polynomial time as discussed earlier. But no vertices sequence should exist which can fool the algorithm to prove the non-hamiltonian graph as Hamiltonian.

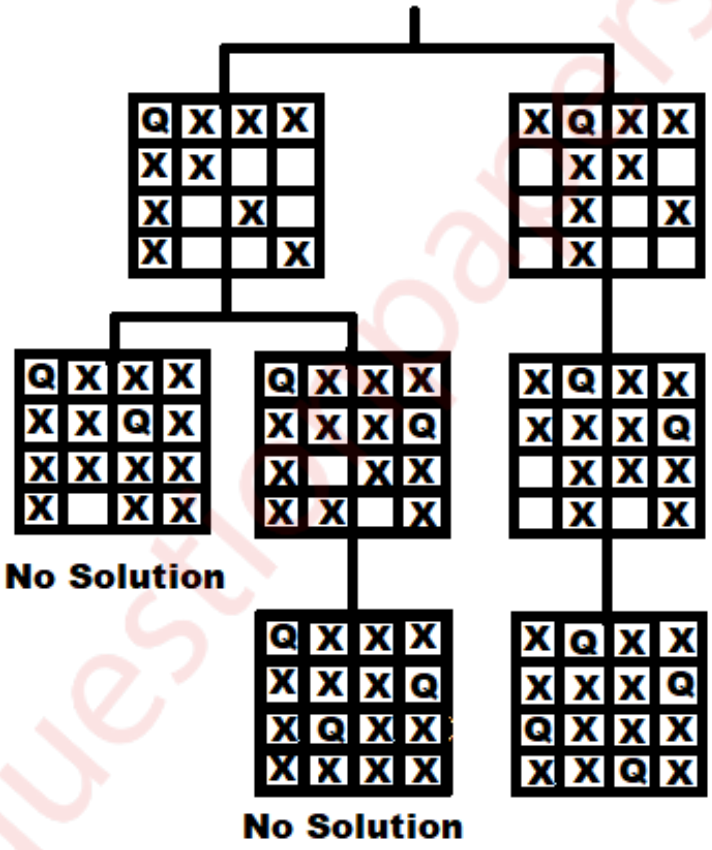
b. **Explain how backtracking is used for solving n- queens problem. Show the state space tree.** 10M

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 6 Write short notes on (any two)** **20M**

**a. Job sequencing with deadlines**.

i) Job sequencing is a problem in which n jobs are arranged in such a way that we get maximum Profit. The job should complete their task within deadlines.

ii) It is also called as Job scheduling with deadlines.

iii) Time complexity = $O(n^2)$

Algorithm

**Step1**: Sort the jobs $J_i$ into non-increasing order.

**Step2**: Assign Empty slot as Maximum deadline value i.e. Empty slot = Dmax.

**Step3**: Take i index value compute value of k

K = min (Dmax, deadline(i))

**Step4**: If value of K is greater and equal to one check empty slot. Empty slot = k

If empty slot is full. check previous slot if it is free assign job to that slot.

If slot is full ignore that job.

**Step5**: increment value of i till all the jobs are not finished. Repeat Step3.

**Step6**: Stop.

Example

Let n=4, (J1, J2, J3, J4)=(100, 10, 15, 27), (D1, D2, D3, D4)=(2, 1, 2, 1) find feasible solution. With job scheduling with deadlines.

| **Index** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Job** | J1 | J2 | J3 | J4 |
| **Value** | 100 | 10 | 15 | 27 |
| **Deadlines** | 2 | 1 | 2 | 1 |

Arrange the jobs in non-increasing value.

| **Index** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Job** | J1 | J4 | J3 | J2 |
| **Value** | 100 | 27 | 15 | 10 |
| **Deadlines** | 2 | 1 | 2 | 1 |

Dmax = 2

| **Time slot** | 1 | 2 |
|---|---|---|
| **Status** | Empty | Empty |

i=1

K= min (Dmax, Deadline(i))

K= min (2, 2)

K=2 ($k \geq 1$)

Time slot = k = 2 (empty)

| Time slot | 1 | 2 |
|---|---|---|
| Status | Empty | J1 |

i=2
K= min (Dmax, Deadline(i))
K= min (2, 1)
K=1 (k ≥ 1)
Time slot = k = 1 (empty)

| Time slot | 1 | 2 |
|---|---|---|
| Status | J4 | J1 |

**Maximum Profit** = J1 + J4
= 27 + 100
= **127**

**b. 8 Queen problem**

i) 8-queens problem is a problem in which 8 queens are arranged 8*8 chess board in such a way that no 2 queens should attack each other.

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

iii) 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. This procedure keeps on going if we don't get feasible solution we backtrack and change the position of previous queen.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| X | X | Q1 | X | X | X | X | X |
| X | X | X | X | X | Q2 | X | X |
| X | Q3 | X | X | X | X | X | X |
| X | X | X | X | X | X | Q4 | X |
| Q5 | X | X | X | X | X | X | X |
| X | X | X | Q6 | X | X | X | X |
| X | X | X | X | X | X | X | Q7 |
| X | X | X | X | Q8 | X | X | X |

iv) 8 queen problem has $^{64}C_8$ = 4, 42, 61, 65, 368 different arrangements, out of these only 92 arrangements are valid solutions. Out of which only 12 are fundamental solution. Rest of 80 solutions can be generated by reflection or rotation.

v) Time Complexity = O(n!)

```
Algorithm Queen (n)
        for column ← 1 to n do
         {
         if (Place(row, column)) then
          {
          Board [row]=column;
          if (row = = n) then
             Print_board (n)
          else
            Queen(row+1, n)
          }
        }

      Place(row, column)
      {
        for i ← row -1 do
          {
          if (board [i] = column) then
            return 0;
          else if (abs(board [i] = column)) = abs(i-row) then
            return 0;
          }
        return 1;
      }
```

**c. Longest common subsequence**

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
```

Example A = ABCF, B = ACF

A =

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| A | B | C | F |

| O | 1 | 2 |
|---|---|---|
| A | C | F |

Draw matrix

| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | | | |
| 0 | | | |
| 0 | | | |
| 0 | | | |

r = 1, c = 1
x [0] = y [0] (true A = A)
LCS [1, 1] = L [0, 0] + 1 = 0+1 = 1

| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | 1 | | |
| 0 | | | |
| 0 | | | |
| 0 | | | |

r = 1, c = 2
x [0] ≠ y [0] (true A = C)
LCS [0, 1] ≥ LCS [1, 1] (0 ≥ 1 (not true))
LCS [1, 2] = LCS [1, 2-1]
LCS [1, 2] = L [1, 1] = 1

| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | 1 | 1 | |
| 0 | | | |
| 0 | | | |
| 0 | | | |

LCS [1, 3] = 1
LCS [2, 1] = 1
LCS [2, 2] = 1
LCS [2, 3] = 1
LCS [3, 1] = 1
LCS [3, 2] = 2
LCS [3, 3] = 2
LCS [4, 1] = 1
LCS [4, 2] = 2
LCS [4, 3] = 3

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 2 | 2 |
| 0 | 1 | 2 | 3 |

LCS =

| 0 | 1 | 2 |
|---|---|---|
| A | C | F |

**********

a) What is backtracking Approach. Explain how it is used in graph coloring.

b) Explain Randomized algorithm with example.

c) What is Knuth Morris Pratt Method of Pattern Matching? Give Examples.

d) Explain in brief the concept of Multistage Graphs?

e) Merge sort and its complixity.

Q2. A) Derive and comment on the complexity of Quick Sort algorithm. 10M

b) Solve Following Knapsack problem using dynamic approach. 10M

N=4 items, capacity of knapsack M= 9

| Item i | Value vi | Weight wi |
|---|---|---|
| 1 | 18 | 2 |
| 2 | 25 | 4 |
| 3 | 27 | 5 |
| 4 | 10 | 3 |

Q3. A) What is sum of Subset problem? Write the Algorithm and solve following. 10M

array A = [2,3,5,6,7,8,9] and K = 15

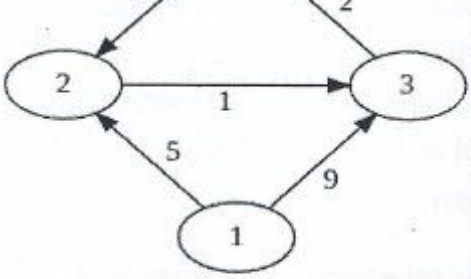
b) Write the algorithm for finding strassens matrix multiplication and show how the complexity is being affected?. 10M

Q4. A) What is Longest Commom subsequence Problem? Find LCS for following. 10M

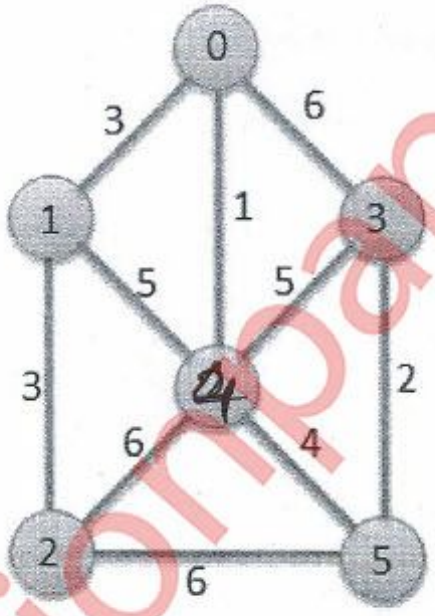
String x = **ACBAED**

String y = ABCABE

b) Explain binary search Tree? How to generate an optimal binary search tree. 10M




b) Find MST of Following Graph using Prims and ~~Prism's~~ Kruskal's Algorithm. 10M




Q6. Write Short Notes on (Any Three) 20M

a) Optimal Storage on Tapes
b) 15 puzzle problem.
c) Binary Search and its complexity.
d) Problem of Multiplying Long Integers.

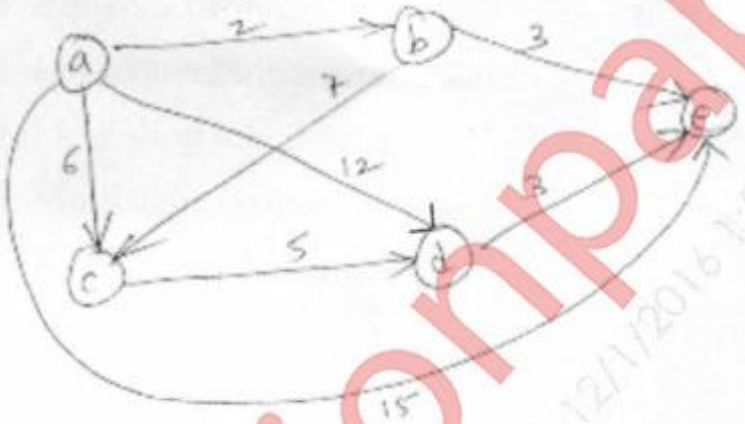
(2) Attempt any three from remaining five questions.

1. (a) Which are the different methods of solving recurrences. Explain v examples.

   (b) Compare Greedy and dynamic programing approach for algorithm Des Explain How both can be used to solve Knapsack problem?

2. (a) Explain the anlaysis of quick sort and apply the same to sort follow data. [ 10 7 5 9 12 3 ]

   (b) Write single source shortest path algorithm & apply the same for follow

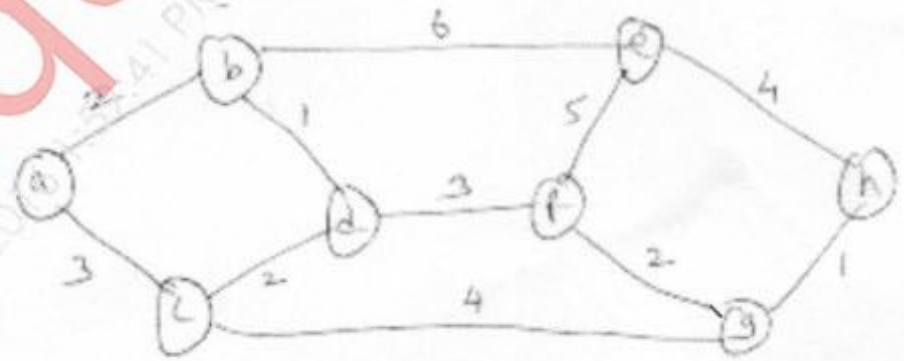



3. (a) Explain string matching with finite automata and apply the same techni to match following pattern.

   txt [ ] = UNIVERSITY OF MUMBAI

   pat [ ] = MBA

   (b) Compare Prims & Kruskal's method for finding Minimum spanning T find MST for following using prims method.




[TURN OVE

search?

(b) Solve sum of subsets problem for following

N = 6 W = { 3, 5, 7, 8, 9, 15 } & M = 20

Also write the Algorithm for it.

5. (a) Explain longest common subsequence problem with example.

(b) What is backtracking method? How it is used in graph coloring problem?

6. Write short notes on **(Any Four)**

(1) 8 queens problem

(2) Job sequencing with deadlines

(3) Flow shop scheduling

(4) Multistage Graphs

(5) A symptotic Notations

---

**Analysis of Algorithm**
**(May 2019)**

**Q.P. Code - 55801**

**Q 1 Solve any 4**

**1. Derive the complexity of quick sort for best case and worst case. 5M**

**Best Case:**

In the best case, every time we partition the array, we divide the list into two nearly equal pieces. Partitions are balanced and it is identical to merge sort. Hence complexity of best case will be O(n $\log_2$ n), Let us prove it by solving recurrence equation.

T(1) = 0

T(n) = 2T(n/2) + θ (n)

2T(n/2): is the time required to solve the problems of size (n/2).

O(n): is the time required to fix the position of the pivot.

Using Master method: T(n) = 2 T($\frac{n}{2}$) + n

Comparing with : T(n) = a T($\frac{n}{b}$) + f(n)

a = 2, b = 2 & f(n) = n with k = 1

where k is degree pf polynomial function f(n)

$a = b^k$, $2 = 2^1$.

So from the case I variant 2

$T(n) = O(n^{\log_2 2} \log n)$

**T(n) = O(n log n)**

"Hence Proved"

**Time Complexity - T(n) = O(n $\log_2$ n)**

Worst Case:

Worst case for quick sort occurs when the list is already sorted. To divide the list, algorithm scans the array and determines the correct position of the pivot, so division cost of Quick Sort is linear, i.e θ (n). In worst case, one of the sub list has size 0. And other has size (n-1).

Thus,

T (0) = 1

T (n) = T(conquer)+T(divide)+ T (combine)

T (n) = T(n-1) + θ (n) + θ (1)

= T (n-1) + n .... (1)

Using iterative approach

Substitute n by (n-1) in equation (1),

$T(n-1) = T(n-2) + (n-1)$ ….(2)

$T(n) = T(n-2) + (n-1) + n$ ….(3)

Substitute n by (n-1) in equation (2)

$T(n-2) = T(n-3) + (n-2)$

From equation (3),

$T(n) = T(n-3) + (n-2) + (n-1) + n$

After k iterations,

$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \ldots + (n-1) + n$

Let k = n,

$T(n) = T(0) + 1 + 2 + 3 + \ldots + (n-2) + (n-1) + (n)$

$= 1 + 2 + 3 + \ldots + n = \sum n$

$= n(n+1)/2 = (n^2/2) + (n/2)$

**$T(n) = O(n^2)$**

**2. What is asymptotic analysis? Define Big O, Omega and Theta notations. 5M**

**Asymptotic Notation:**

Asymptotic analysis are a mathematical tool to find time or space complexity of an algorithm without implementing it in a programming language. It is a way of describing a major component of the cost of the entire algorithm.

- Asymptotic notation does the analysis of algorithm independent of all such parameters

**Big O:**

Let f(n) and g(n) are two nonnegative functions indicating running time of two algorithms. We say, g(n) is upper bound of f (n) if there exist some positive constants c and $n_o$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_o$. It is denoted as f (n) = O(g(n)).

- The Big O notation, where O stands for 'order of', is concerned with what happens for very large values of n.
- The Big O notation defines upper bound for the algorithm, it means the running time of algorithm cannot be more than its asymptotic upper bound for any random sequence of data

**Big omega:**

Let f(n) and g(n) are two nonnegative functions indicating running time of two algorithms. We say, g(n) is lower bound of function f (n) if there exist some positive constants c and $n_o$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_o$. It is denoted as f (n) = Ω(g(n)).

- This notation is denoted by 'Ω', and it is pronounced as "Big Omega".
- Big Omega notation defines lower bound for the algorithm.

**Big Theta:**

Let f(n) and g(n) are two nonnegative functions indicating running time of two algorithms. We say, g(n) is tight bound of function f (n) if there exist some positive constants c1, c2 and $n_o$ such that 0 ≤ c1. g(n) ≤ f(n) ≤c2. g(n) for all n ≥ $n_o$. It is denoted as f (n) = θ(g(n)).

- This notation is denoted by 'θ' and it is pronounced as Big theta.
- Big theta defines tight bound for the algorithm.

**3. Write an algorithm to find all pairs shortest path using dynamic programming. 5M**

Algorithm for all pair shortest path:

```
for all vertices u
 for all vertices v
 if u = v
dist [u, v] ← 0
 else
dist [u, v] ←∞
 for l← 1 to V - 1
 for all vertices u
 for all edges x → v
 if dist [u, v] > dist [u, x] + w(xv)
dist [u, v] ← dist [u, x] + w(xv)
```

**4. Write a note on " Optimal Storage on Tapes". 5M**

Problem:

Given n programs P1, P2, …, Pn of length L1, L2, …, Ln respectively, store them on tap of length L such that Mean Retrieval Time (MRT) be minimum.

Retrieval time of the $j^{th}$ program is a summation of the length of first j programs on tap. Let $T_j$ be the time to retrieve program $P_j$. Retrieval time of $P_j$ is computed as,

$$T_j = \sum_{k=1}^{j} L_k$$

Mean retrieval time of n programs. It is required to store programs in an order such that their Mean Retrieval Time is minimum.

Optimal Storage on tape is minimization problem which,

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{k=1}^{i} L_k$$

$$\text{Subjected to} \sum_{i=1}^{n} L_i \leq L$$

- In case we have to find the permutation of the program order which minimizes the MRT after storing all programs on single tape only.
- Greedy algorithm stores the programs on tape in nondecreasing order of their length, which ensures the minimum MRT.

- A magnetic tape provides only sequential access of data, In an audio tape, cassette, unlike a CD, a fifth song from the tape cannot be just directly played. The length of the first four songs must be traversed to play the fifth song. So in order to access certain data, head of the tape should be positioned accordingly.

5. **Define master theorem. Solve the following using master method $T(n) = 8T(n/2) + n^2$** **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 $T(n) = a \cdot T(n/b) + f(n)$. 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) =** $\left(n^{log_b^a}\right)$ **=** $\boldsymbol{\theta}\left(n^{log_2^8}\right)$ **=** $\boldsymbol{\theta}\left(n^{log_2^{2^3}}\right)$ **=** $\boldsymbol{\theta}\left(n3^{log_2^2}\right)$ **=** $\boldsymbol{\theta}\,(n^3)$

---

**Q 2**

**A) write an algorithm for finding minimum and maximum using divide and conquer. Also derive its complexity.** **10M**

Algorithm: **Algorithm DC_MAXMIN (A, Low, high)**
**If n = = 1 then**

```
return (A[1], A[1])
else if n = = 2 then
if A[1] < A[ 2] then
return (A[1], A[2])
else
return (A[2], A[1])
else
mid ← (low + high)/2
[LMin, LMax]= DC_MAXMIN(A, low,mid)
[RMin, RMax]= DC_MAXMIN(A, mid + 1, high)
If LMax > RMax then
Max ← LMax
Else
Max ← RMax
End
If LMin < RMin then
min← LMin
else
min←RMin
end
return (min, max)
end
```

Complexity of algorithm:

- DC_MAXMIN does two comparisons two determine minimum and maximum element and creates two problems of size n/2, so the recurrence can be calculated as,

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ 2T(n/2) + T(n/2) + 2 & n>2 \end{cases}$$

When n is a power of two, $n = 2^k$
for some positive integer k, then

$T(n) = 2T(n/2) + 2$
$= 2(2T(n/4) + 2) + 2$
$= 4T(n/4) + 4 + 2$
.
.
.
$= 2^{k-1} T(2) + \sum(1 \le i \le k-1)\ 2^k$
$= 2^{k-1+} 2^k - 2$
$= 3n/2 - 2 = O(n)$

**3n/2 – 2 is the best, average, worst case number of comparison when n is a power of two.**

**B) write Kruskal's algorithm and show it's working by taking suitable example of graph with 5 vertices.**

**10M**

Algorithm KRUSKAL_MST(G)

A = Ø

Foreach v ∈ G . V:

MAKE-SET (v)

Foreach ( u, v) in G . E ordered by weight (u, v), increasing:

if FIND-SET (u) ≠ FIND-SET(V):

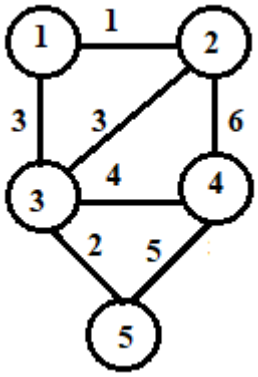
A = A ∪ {(u, v)}

UNION(FIND-SET(u), FIND-SET(v))

return A

Example:

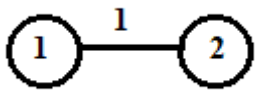
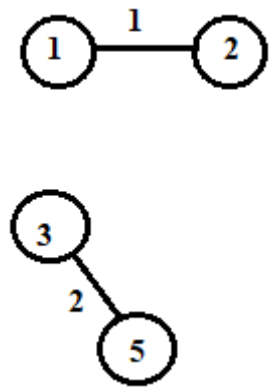
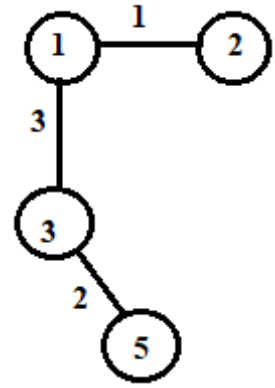
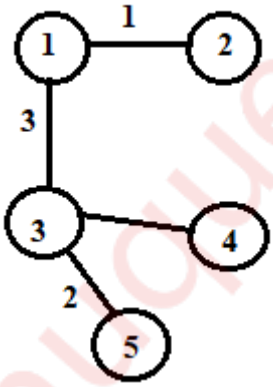
Find the cost of minimal spanning tree of the given graph by using Kruskal's Algorithm.




Solution:

| Edge | <1,2> | <3,5> | <1,3> | <2,3> | <3,4> | <4,5> | <2,4> |
|---|---|---|---|---|---|---|---|
| Cost | 1 | 2 | 3 | 3 | 4 | 5 | 6 |

- Kruskal's algorithm sorts the edges according to decreasing order of their weight. Sorted edges are listed in following table.
- One by one edge is added to partial solution if it is not forming the cycle.
- Initially, set of unvisited edges $U_E$ = { <1,2>, <3,5>, <1,3>, <2,3>, <3,4>, <4,5>, <2,4>}

| Partial solution | Updated $U_E$ |
|---|---|
|  | $U_E$ = {<3,5>, <1,3>, <2,3>, <3,4>, <4,5>, <2,4>} |
|  | $U_E$ = {<1,3>, <2,3>, <3,4>, <4,5>, <2,4>} |
|  | $U_E$ = {<2,3>, <3,4>, <4,5>, <2,4>} |
|  | $U_E$ = {<4,5>, <2,4>} |
| Minimum cost edge <2,4> creates cycle so remove it from $U_E$ | **Cost of solution:**<br>**w(1,2)+w(1,3)+W(3,4)+w(3,5)**<br>**=1+3+4+2 = 10** |

**Q 3**

**A) Solve fractional knapsack problem for the following.**

**n = 6, p= (18, 5, 9, 10, 12, 7) w = (7, 2, 3, 5, 3, 2), Max sack capacity M = 13.** **10M**

Solution:

| Item | Weight | Value | Value / Weight |
|---|---|---|---|
| P1 | 7 | 18 | 2.571 |
| P2 | 2 | 5 | 2.5 |
| P3 | 3 | 9 | 3 |
| P4 | 5 | 10 | 2 |
| P5 | 3 | 12 | 4 |
| P6 | 2 | 7 | 3.5 |

Arrange the Item in increasing order of value / Weight Ratio

P5, P6, P3, P1, P2, P4

Capacity of Bag is 13.

| Capacity of Bag | Items | value |
|---|---|---|
| 13 | ----- | 0 |
| 10 | P5 | 12 |
| 8 | P5, P6 | 7 |
| 5 | P5, P6, P3 | 9 |

Now, P1 arrives but capacity of bag is only 5 and weight is 7 so we are going to take fraction

**Maximum Profit** = $28 + \frac{5}{7} * 18$ **= 40.852**

**B) Write an algorithm for Knuth Morris Pratt (KMP) pattern matching.** **10M**

i) 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.

ii) Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive approach.

iii) This algorithm builds a prefix array. This array is also called as ∏ array.

iv) Prefix array is build using prefix and suffix information of pattern.

v) 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]
```

```
        q = 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 ∏
```

---

**Q 4**

**A) Write an algorithm to solve N Queens problem. Show its working for N = 4.** **10M**

```
Algorithm Queen (n)
        for column ← 1 to n do
         {
         if (Place(row, column)) then
          {
          Board [row]=column;
          if (row = = n) then
             Print_board (n)
          else
            Queen(row+1, n)
          }
        }

      Place(row, column)
      {
      for i ← row -1 do
        {
          if (board [i] = column) then
            return 0;
          else if (abs(board [i] = column)) = abs(i-row) then
            return 0;
        }
```

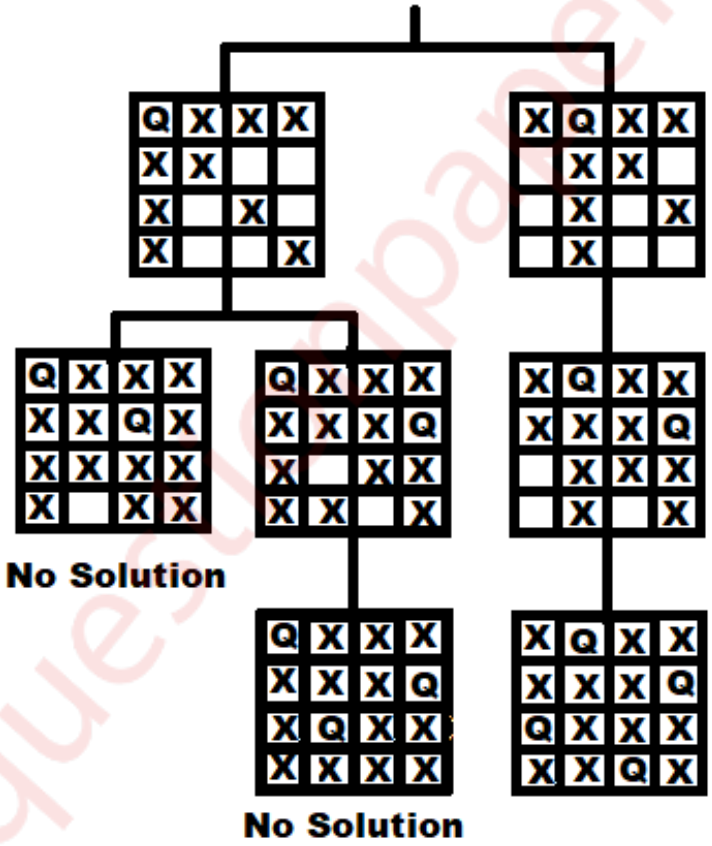
```
    return 1;
}
```

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.

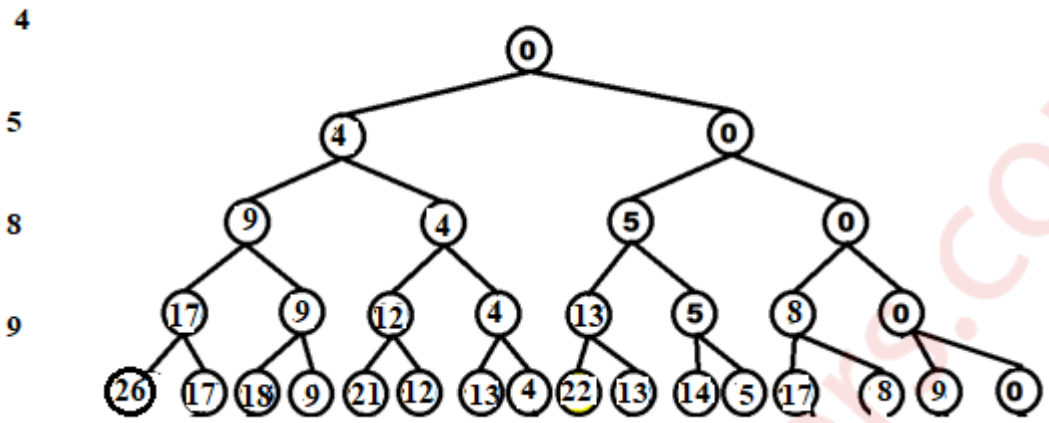
iv) In a similar way, if (1, 1) position is not feasible for queen 1, then algorithm backtracks and put the first queen cell (1, 2), and repeats the procedure. For simplicity, only a few nodes are shown below in state space tree.




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.

**B) Write an algorithm to solve sum of subset problem and solve the following problem. n =4, w = {4, 5, 8, 9}, required sum = 9.** **10M**




**Solutions** : Subset **1) {4, 5}**

**Algorithm:**

Let W be a set of elements & M be the expected sum of subset then

**Step1**: Start with empty set.

**Step2**: Add to the subset, the next element from the list.

**Step3**: If the subset is having sum equal to M then Stop with that subset as solution,

**Step4**: If the subset is not matching with the M or if we have reached to the end of the set Then backtrack through that subset until we find suitable value.

**Step5**: If the subset is less then M then repeat Step2.

**Step6**: If we have visited all the elements without finding suitable & No backtrack is possible Then stop without solution.

Time Complexity – $O(2^n)$

---

**Q 5**

**A) Prove that Vertex Cover problem is NP Complete.** **10M**

Given that the Independent Set (IS) decision problem is N P-complete, prove that Vertex Cover (VC) is N P-complete.

Solution: 1. Prove Vertex Cover is in N P. I Given VC ,

vertex cover of G = (V, E), |VC | = k I We can check in O(|E| + |V|) that VC is a vertex cover for G.

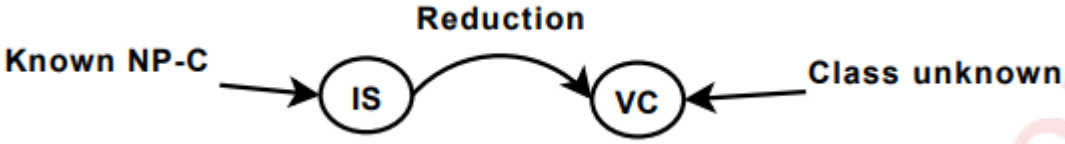
For each vertex ∈ VC , remove all incident edges.

Check if all edges were removed from G.

Thus, Vertex Cover ∈ N P

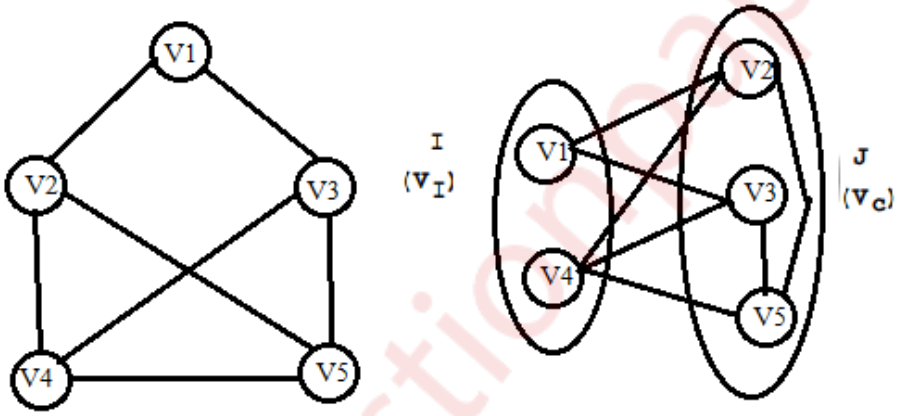
Select a known N P-complete problem.

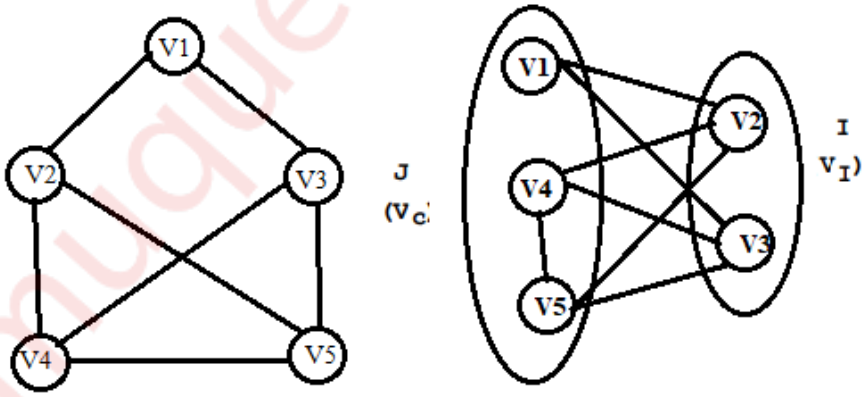
- Independent Set (IS) is a known N P-complete problem.
- Use IS to prove that VC is N P-complete.




3. 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.



**B) Find the longest common subsequence for the following two strings.**

**X = ABACABBY = BABCAB** **10M**

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.

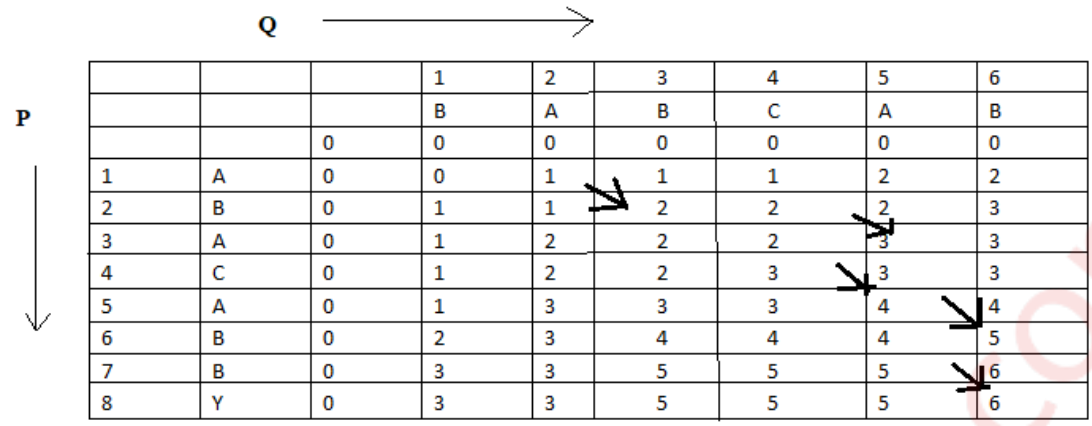
iii) This is also one type of string-matching technique. It works on brute force approach.

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
```

**Let us consider P=( A, B, A, C, A, B, B, Y) and Q=( B, A, B, C, A, B)**




| | | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| | | | B | A | B | C | A | B |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | 0 | 1 | 1 | 1 | 2 | 2 |
| 2 | B | 0 | 1 | 1 | 2 | 2 | 2 | 3 |
| 3 | A | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| 4 | C | 0 | 1 | 2 | 2 | 3 | 3 | 3 |
| 5 | A | 0 | 1 | 3 | 3 | 3 | 4 | 4 |
| 6 | B | 0 | 2 | 3 | 4 | 4 | 4 | 5 |
| 7 | B | 0 | 3 | 3 | 5 | 5 | 5 | 6 |
| 8 | Y | 0 | 3 | 3 | 5 | 5 | 5 | 6 |

**LCS : (A,B, C,Y)**

---

**Q 6**

**A) Assembly Line Scheduling** **10M**

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

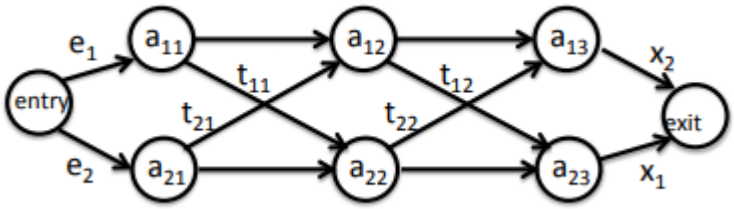
- Manufacturing of large items like car, trucks etc. generally undergoes through multiple stations, where each station is responsible for assembling particular part only. Entire product be ready after it goes through predefined n stations in sequence.

- 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.




**B) Job Sequencing with deadlines** **10M**

i) Job sequencing is a problem in which n jobs are arranged in such a way that we get maximum Profit. The job should complete their task within deadlines.

ii) It is also called as Job scheduling with deadlines.

iii) Time complexity = $O(n^2)$

Algorithm

**Step1**: Sort the jobs $J_i$ into non-increasing order.

**Step2**: Assign Empty slot as Maximum deadline value i.e. Empty slot = Dmax.

**Step3**: Take i index value compute value of k

K = min (Dmax, deadline(i))

**Step4**: If value of K is greater and equal to one check empty slot. Empty slot = k

If empty slot is full. check previous slot if it is free assign job to that slot.

If slot is full ignore that job.

**Step5**: increment value of i till all the jobs are not finished. Repeat Step3.

**Step6**: Stop.

Example

Let n=4, (J1, J2, J3, J4)=(100, 10, 15, 27), (D1, D2, D3, D4)=(2, 1, 2, 1) find feasible solution. With job scheduling with deadlines.

| **Index** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Job** | J1 | J2 | J3 | J4 |
| **Value** | 100 | 10 | 15 | 27 |
| **Deadlines** | 2 | 1 | 2 | 1 |

Arrange the jobs in non-increasing value.

| **Index** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Job** | J1 | J4 | J3 | J2 |
| **Value** | 100 | 27 | 15 | 10 |
| **Deadlines** | 2 | 1 | 2 | 1 |

Dmax = 2

| **Time slot** | 1 | 2 |
|---|---|---|
| **Status** | Empty | Empty |

i=1
K= min (Dmax, Deadline(i))
K= min (2, 2)
K=2 (k ≥ 1)
Time slot = k = 2 (empty)

| **Time slot** | 1 | 2 |
|---|---|---|
| **Status** | Empty | J1 |

i=2
K= min (Dmax, Deadline(i))
K= min (2, 1)
K=1 (k ≥ 1)
Time slot = k = 1 (empty)

| **Time slot** | 1 | 2 |
|---|---|---|
| **Status** | J4 | J1 |

**Maximum Profit** = J1 + J4
= 27 + 100
= **127**

**C) 15 Puzzle Problem** **10M**

- In this problem there are 15 tiles, which are numbered from 0 – 15.

- The objective of this problem is to transform the arrangement of tiles from initial arrangement to a goal arrangement.

- 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)

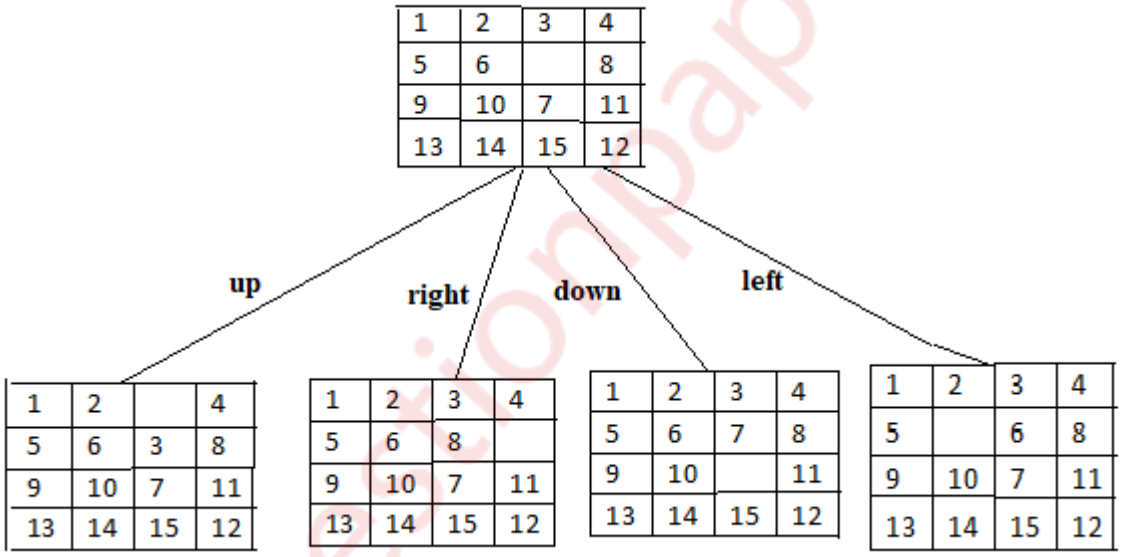



**Fig. Cost f(x) + h(x) after first move**

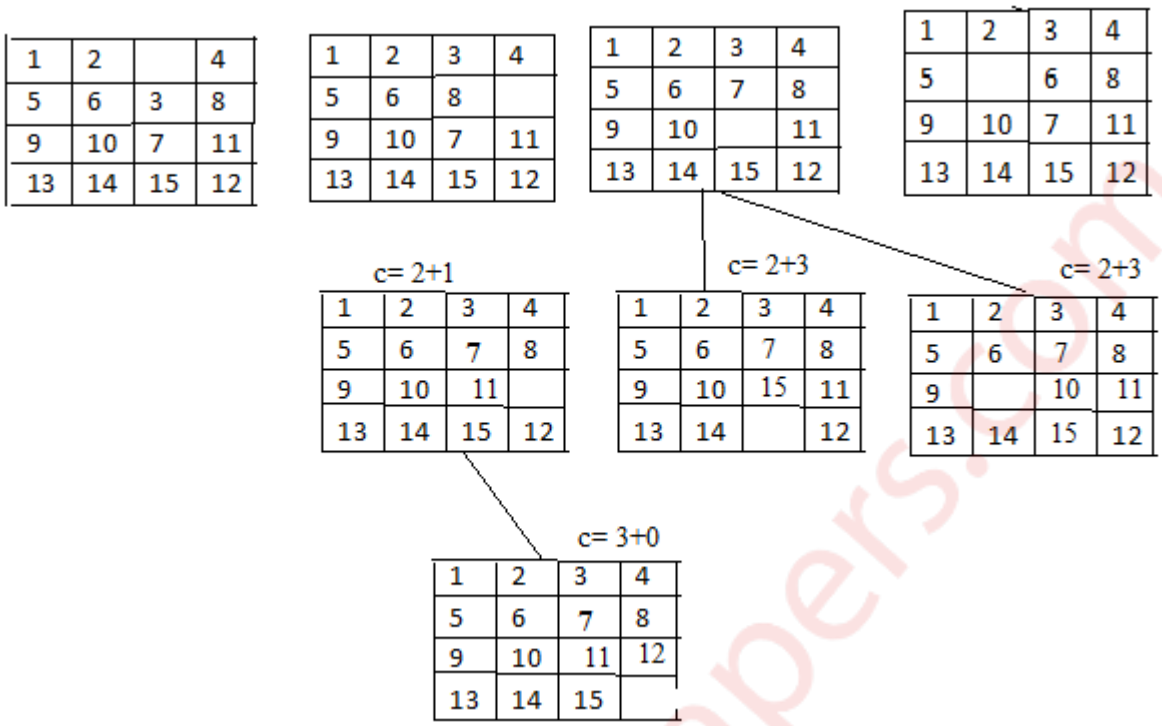



**Fig. Tile positions after subsequent moves**

**D) P, NP, and NPC Classes** **10M**

**P:**

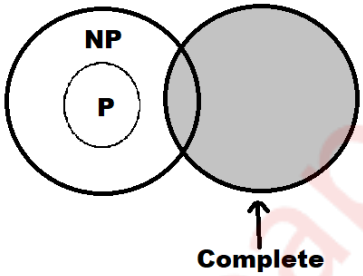
- Polynomial time solving .
- Problems which can be solved in polynomial time, which take time like O(n), O(n2), O(n3).
- Example: finding maximum element in an array or to check whether a string is palindrome or not.
- so there are many problems which we can solved in polynomial time.
- P problems are 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.




NP Complete

---

**********

**Analysis of Algorithm**
**(DEC 2018)**

**Q.P. Code – 55800**

**Q 1**

**A) Explain Strassen's matrix multiplication concept with an example, derive it's complexity .** **10M**

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

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

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

.
.

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

Let's assume $n = 2^k$

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

$$= n^{log_2 7} = n^{2.81} < n^3$$

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

**B) Apply the quick sort algorithm to sort the list E,X,A,M,P,L,E in alphabetical order. Analyze the best case, worst case and average case and complexities of quick sort. 10M**

**Example:**

Given List:

| E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Step1:

| E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|

pivot low high

| E | E | A | M | P | L | X |
|---|---|---|---|---|---|---|

Pivot low high

| E | E | A | M | P | L | X |
|---|---|---|---|---|---|---|

Pivot low high

| E | E | A | M | P | L | X |
|---|---|---|---|---|---|---|

pivot low high

| E | E | A | L | P | M | X |
|---|---|---|---|---|---|---|

pivot Low high

| E | E | A |
|---|---|---|

Left partition

| P | M | X |
|---|---|---|

Right partition

| L |
|---|

pivot

SO, the final list will be

| **A** | **E** | **E** | **L** | **M** | **P** | **X** |
|---|---|---|---|---|---|---|

**Complexities:**

**- Worst-case**: The worst-case behavior for quicksort occurs when the partitioning routine produces one subproblem with n−1 elements and one with 0 elements.

- The partitioning costs $\theta$(n) time.
- T(n) = T(n−1)+T(0)+ $\theta$ (n)
  = T(n−1)+$\theta$ (n).
  T(n) = $\theta(n^2)$
- Therefore the worst-case running time of quicksort is no better than that of insertion sort.

**- Best-case**: In the most even possible split, PARTITION produces two subproblems, each of size no more than n/2, since one is of size(n/2)and one of size(n/2)−1.
- T(n) ≤ 2T(n/2)+$\theta$ (n)
- T(n) = O(nlgn)

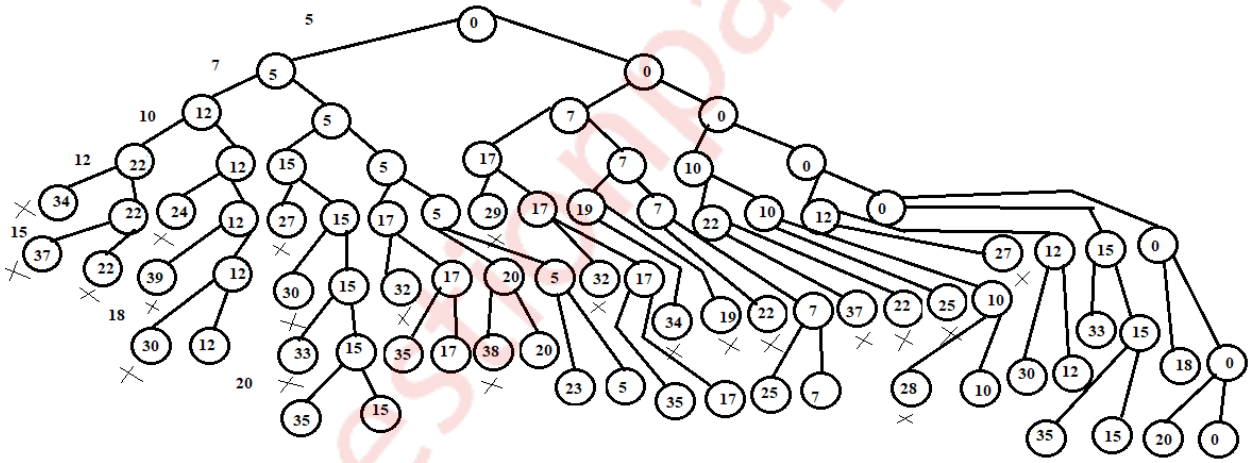
**- Average-case:** The average-case running time of quicksort is much closer to the best case than to the worst case.
- In the average case, PARTITION produces a mix of "good" and "bad" splits. In a recursion tree for anaverage-case execution of PARTITION, the good and bad splits are distributed randomly throughout the tree.
- O(nlgn).\

---

**Q 2**

**A) Solve the following problem of sum of subset and draw portion of state space tree. W= (5, 7, 10, 12, 15, 18, 20) m=35**

**Find all possible subset of w that sum to m.** **10M**



Solution:

1) (5,8,12)
2) (15,20)
3) (5,10,20)
4) (5,12,18)

**B) What is single source shortest path algorithm. Write an algorithm to find single source path using greedy methods.** **10M**

- In a shortest-paths problem a weighted, directed graph G = (V, E), with weight function w : E → R mapping edges to real-valued weights.
- The weight of path p =( v0, v1, ..., vk) is the sum of the weights of it's constituent edges

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Bellman-Ford algorithm is used to find the shortest path from the source vertex to remaining all other vertices in the weighted graph.
- It is slower compared to Dijkstra algorithm but it can handle negative weights also.
- If the graph contains negative-weight cycle, it is not possible to find the minimum path, because on every iteration of cycle gives a better result.
- Bellman-Ford algorithm can detect a negative cycle in the graph but it cannot find the solution for such graphs.
- The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.
- Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.

**- Algorithm:**

**//Initialization**
**for each v ϵ V do**
**d[v]← ∞**
**π[v]← NULL**
**end**
**d[s]←0**
**//Relaxation**
**i←1 to |V| - 1 do**
**for each edge (u, v) ϵ E do**
**if d[u] + w (u,v) <d[v] then**
**d[v]← d[u] + w(u, v)**
**π[v]← u**
**end**
**end**
**end**
**//check for negative cycle**
**For each edge (u, v) ϵ E do**
**If d[u] +w(u, v)<d[v] then**
**Error "graph contains negative cycle"**
**end**
**end**
**return d, π**

---

**Q 3**

**A) Prove that vertex cover problem is NP complete.** **10M**

Given that the Independent Set (IS) decision problem is N P-complete, prove that Vertex Cover (VC) is N P-complete.

Solution: 1. Prove Vertex Cover is in N P. I Given VC ,

vertex cover of G = (V, E), |VC | = k I We can check in O(|E| + |V|) that VC is a vertex cover for G.

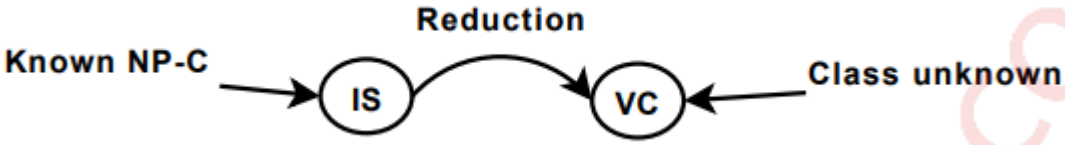
For each vertex ∈ VC , remove all incident edges.

Check if all edges were removed from G.

Thus, Vertex Cover ∈ N P

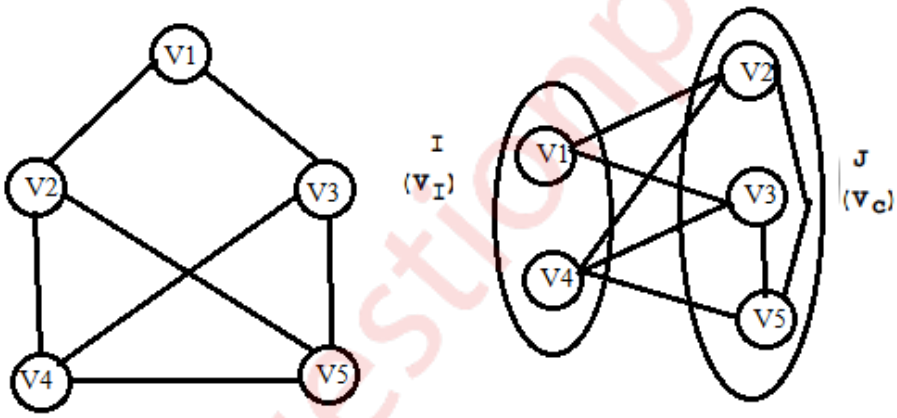
Select a known N P-complete problem.

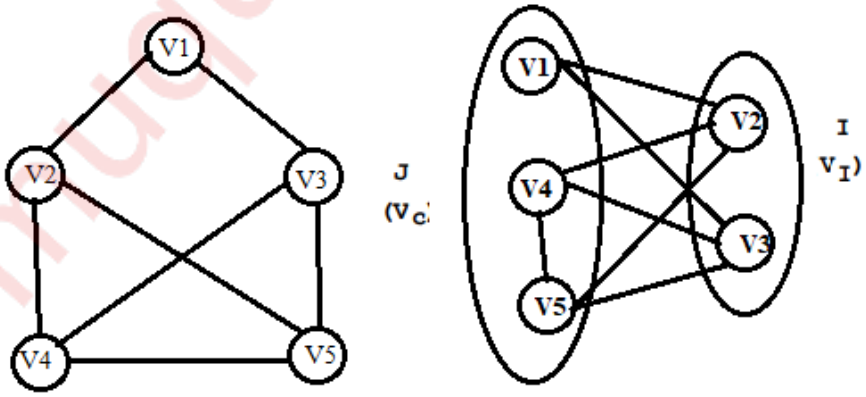
- Independent Set (IS) is a known N P-complete problem.
- Use IS to prove that VC is N P-complete.




3. 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.



**B) Explain various String matching algorithm.** **10M**

There are various String matching algorithms listed below.

**A] Naive:**

i) It is the simplest method which uses brute force approach.

ii) It is a straight forward approach of solving the problem.

iii)It compares first character of pattern with searchable text. If match is found, pointers in both strings are advanced. If match not found, pointer of text is incremented and pointer of pattern is reset. This process is repeated until the end of the text.

iv)It does not require any pre-processing. It directly starts comparing both strings character by character.

v) Time Complexity = O(m*(n-m))

Algorithm -- **NAVE_STRING_MATCHING (T, P)**

```
for i  0 to n-m do
if P[1…….m] = = T[i+1…..i+m] then
print “Match Found”
end
end
```

**B] Rabin-Karp:**

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) Finite Automata:**

**i)** Idea of this approach is to build finite automata to scan text T for finding all occurrences of of pattern P.

**ii)** This approach examines each character of text exactly once to find the pattern. Thus it takes linear time for matching but preprocessing time may be large.

**iii)** It is defined by tuple M = {Q, ∑, $q_o$, F, ∂}

Where Q = Set of States in finite automata

∑ = Sets of input symbols

$q_o$ = Inital state

F = Final State

∂ = Transition function iv) Time Complexity = O($M^3$|∑|)

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

```
State ← 0
for I ← 1 to n
State ← ∂(State, t_i)
If State == m then
Match Found
end
end
```

**D) Knuth Morris Pratt (KMP)**

**i)** 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.

**ii)** Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive approach.

**iii)** This algorithm builds a prefix array. This array is also called as ∏ array.

**iv)** Prefix array is build using prefix and suffix information of pattern.

**v)** This algorithm achieves the efficiency of O(m+n) which is optimal in worst case.

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

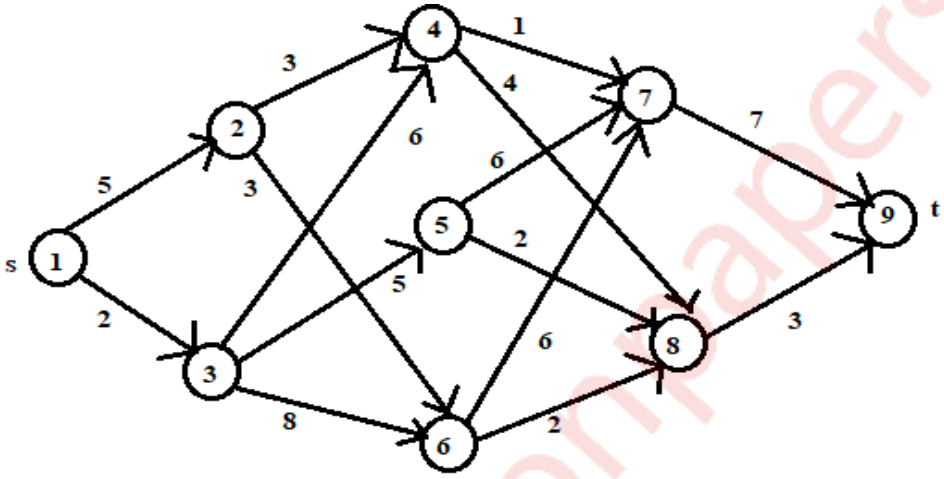
```
n = T.length
m = P.length
∏ = Compute prefix
q  0
for i = 1 to n
while q > 0 and P[q+1] ≠ T[i]
q = ∏ [q]
if P[q+1] = = T[i]
p= q+1
if q = = m
Print “pattern found”
q = ∏ [q]
COMPUTE_PREFIX (P)
M = P.length
Let ∏ [1……m] be a new array
```

**∏ [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 ∏**

---

**Q 4**

**A) Find the minimum cost path from s to t in the following figure using multistage graph. 10M**




**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

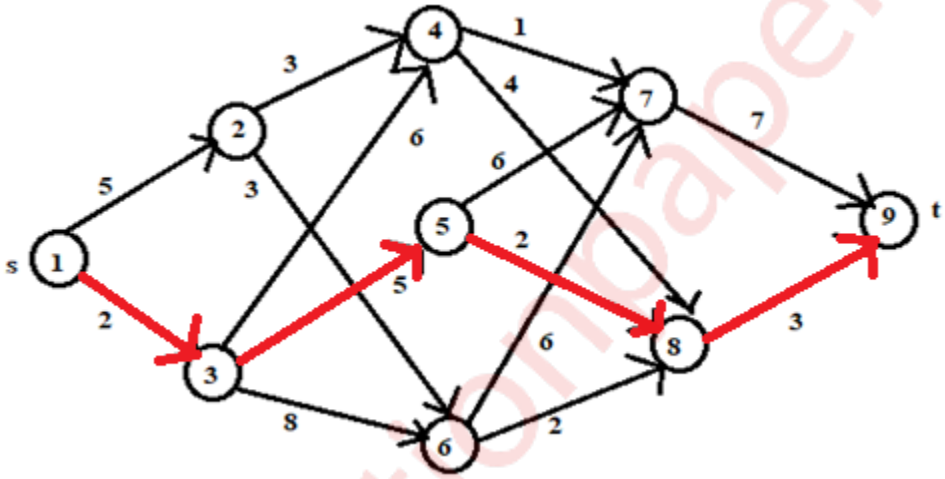
**Stage 3:**
Vertex 4 is connected 7 and 8
Cost [4] = min{c[4,7], c[4,8]}
=min {1, 4}
= 1

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**




**B) Describe the travelling sales person problem and discuss how to solve it using dynamic programming with example.** **10M**

- In the traveling-salesman problem given a complete undirected graph G = (V, E) that has a nonnegative integer cost c(u,v) associated with each edge (u,v) ∈ E, and we must find a Hamiltonian cycle (a tour) of G with minimum cost.
- As an extension of our notation, let c(A) denote the total cost of the edges in the subset A ⊆ E

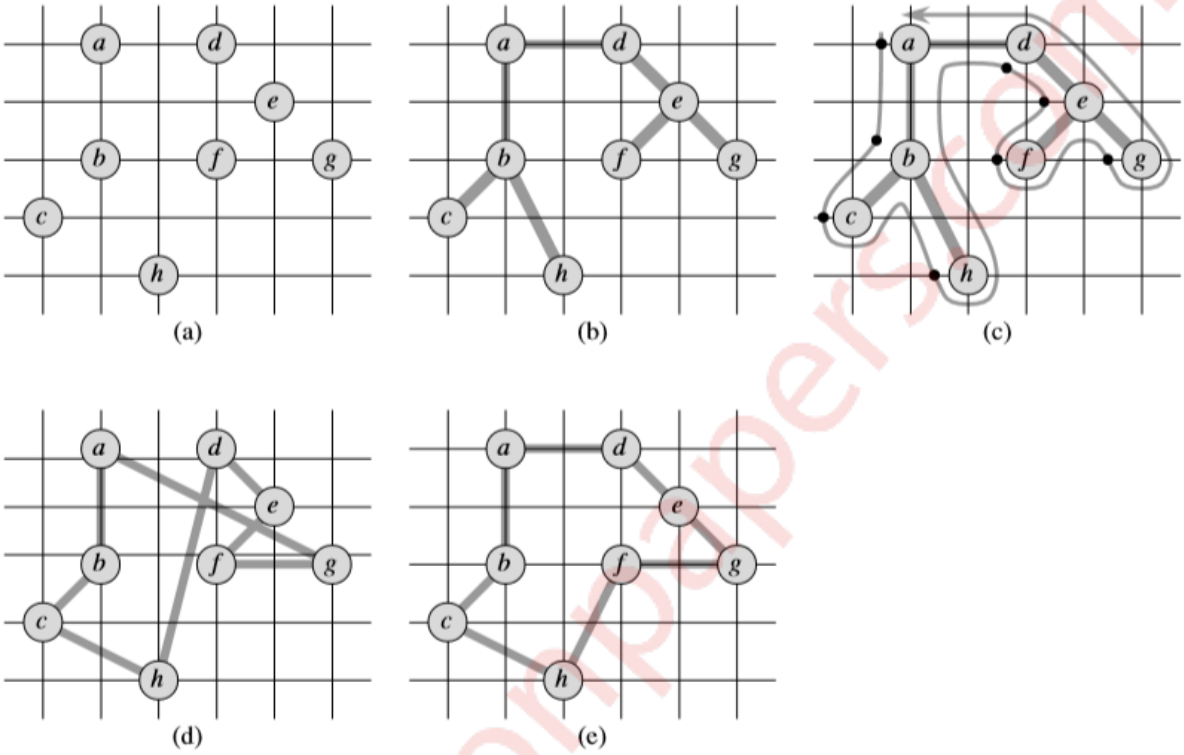
$$c(A) = \sum_{(u,v)\epsilon A} c(u,v)$$

- We formalize this notion by saying that the cost function c satisfies the triangle inequality if for all vertices u,v,w∈ V,
- c(u,w)≤c(u,v)+c(v,w)
- The triangle inequality is a natural one and in many applications it is automatically satisfied
- Algorithm for travelling sales man problem:

APPROX-TSP-TOUR(G,c)
1 select a vertex r ∈ V[G] to be a“root” vertex

2 compute a minimum spanning tree T for G from root r using MST-PRIM(G,c,r)
3 let L be the list of vertices visited in a preorder tree walk of T
4 return the hamiltonian cycle H that visits the vertices in the order L

Example:




---

**Q 5**

**A) What is longest common subsequence problem? find the LCS for the following problem.**

**S1= abcdaf**

**S2= acbcf** **10M**

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
```

**Example:**

S1= abcdaf

S2= acbcf

Formula:

$$LCS(i,j) = \begin{cases} 0 & ,if\ i = 0\ or\ j = 0 \\ 1 + LCS[i-1, j-1] & ,if\ \ p_i = Q_j \\ \max(LCS[i, j-1], LCS[i-1, j]) & ,p_i \neq Q_j \end{cases}$$

LCS[1,1] ➔ i= 1, j=1, $p_i = A$, $Q_i$=B

LCS[1,1]=1 LCS[2,1]=1 LCS[3,1]=1 LCS[4,1]=1 LCS[5,1]=2

LCS[1,2]=1 LCS[2,2]=1 LCS[3,2]=2 LCS[4,2]=2 LCS[5,2]=2

LCS[1,3]=1 LCS[2,3]=1 LCS[3,3]=2 LCS[4,3]=3 LCS[5,3]=2

LCS[1, 4]=1 LCS[2,4]=1 LCS[3,4]=2 LCS[4,4]=3 LCS[5,4]=2

LCS[1,5]=2 LCS[2,5]=2 LCS[3,5]=2 LCS[4,5]=3 LCS[5,5]=2

LCS[1,6]=2 LCS[2,6]=3 LCS[3,6]=2 LCS[4,6]=3 LCS[5,6]=2

$p_i$ $Q_i$ →

| | | | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|---|---|
| | | | **a** | **b** | **c** | **d** | **a** | **F** |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | **a** | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| **2** | **c** | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| **3** | **b** | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| **4** | **c** | 0 | 1 | 2 | 3 | 3 | 3 | 3 |
| **5** | **f** | 0 | 1 | 2 | 3 | 3 | 3 | 4 |

S1= abcdaf

S2= acbcf

So, LCS = abcf

**B) Write short note on 8 queen problem, write an algorithm for the same. 10M**

i) 8-queens problem is a problem in which 8 queens are arranged 8*8 chess board in such a way that no 2 queens should attack each other.

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

iii) 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. This procedure keeps on going if we don't get feasible solution we backtrack and change the position of previous queen.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| X | X | Q1 | X | X | X | X | X |
| X | X | X | X | X | Q2 | X | X |
| X | Q3 | X | X | X | X | X | X |
| X | X | X | X | X | X | Q4 | X |
| Q5 | X | X | X | X | X | X | X |
| X | X | X | Q6 | X | X | X | X |
| X | X | X | X | X | X | X | Q7 |
| X | X | X | X | Q8 | X | X | X |

iv) 8 queen problem has ${}^{64}C_8$ = 4, 42, 61, 65, 368 different arrangements, out of these only 92 arrangements are valid solutions. Out of which only 12 are fundamental solution. Rest of 80 solutions can be generated by reflection or rotation.

v) Time Complexity = O(n!)

Algorithm **Queen (n)**

```
for column ← 1 to n do
{
if (Place(row, column)) then
{
Board [row]=column;
if (row = = n) then
Print_board (n)
else
Queen(row+1, n)
}
}
Place(row, column)
```

```
{
for i ← row -1 do
{
if (board [i] = column) then
return 0;
else if (abs(board [i] = column)) = abs(i-row) then
return 0;
}
```

---

**Q 6 write a short note.**

**A) Branch and Bound strategy** **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.
- Application: Travelling salesman problem, Knapsack problem

-Branch and bound technique generate all the child nodes of E-node before another node becomes E node.

-Let c(x) represent the cost of answer node x. The aim is to find minimum cost answer node.

-Search strategies like FIFO, LIFO and LC search differs in terms of the sequence in which they explore the node in state space tree.

Branch and bound method employ either BFS or DFS search. During BFS, expanded nodes are kept in a queue, whereas in DFS nodes are kept on the stack.

- Example: FIFO branch and bound approach.

**B) Algorithms to find minimum spanning tree** **10M**

- A spanning tree of a connected undirected graph G, is a sub-graph of G which is a tree that connects all the vertices together.
- A graph G can have many different spanning trees. We can assign weights to each edge.

- Use it to assign a weight to a spanning tree by calculating the sum of the weights of the edges in that spanning.
- A minimum spanning tree (MST) is defined as a spanning tree with weight less than or equal to the weight of every other spanning tree.

**- Algorithms:**

**Prim's algorithm:**

- Prim's algorithm is a greedy algorithm that used to form a minimum spanning tree for a connected weighted undirected graph. In other words, the algorithm builds a tree that includes every vertex and a subset of the edges in such a way that the total weight of all the edges in the tree is minimized.
  - Tree vertices: Vertices that are a part of the minimum spanning tree T.
  - Fringe vertices: Vertices that are currently not a part of T, but are adjacent to some tree vertex.
  - Unseen vertices: Vertices that are neither tree vertices nor fringe vertices fall under this category.

- The steps involved in the Prim's algorithm:

**Step 1:** Select a starting vertex
**Step 2:** Repeat Steps3and4until there are fringe vertices
**Step 3:** Select an edge connecting the tree vertex and fringe vertex that has minimum weight
**Step 4:** Add the selected edge and the vertex to the minimum spanning tree T [END OF LOOP]
**Step 5:** EXIT

**Kruskal's algorithm:**

- Kruskal's algorithm is used to find the minimum spanning tree for a connected weighted graph.
- The algorithm aims to find a subset of the edges that forms a tree that includes every vertex.
- The total weight of all the edges in the tree is minimized.
- However, if the graph is not connected, then it finds a minimum spanning forest.
- Kruskal's algorithm is an example of a greedy algorithm, as it makes the locally optimal choice at each stage with the hope of finding the global optimum.
- Algorithm:

**Step 1:** Createaforest in suchaway that each graph isaseparate tree.
**Step 2:** Createapriority queueQthat contains all the edges of the graph.
**Step 3:** Repeat Steps4and5whileQis NOT EMPTY
**Step 4:** Remove an edge from Q
**Step 5:** IF the edge obtained in Step4connects two different trees, then Add it to the forest (for combining two trees into one tree). ELSE Discard the edge
**Step 6:** END

**C) Recurrences** **10M**

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 $n = 0$

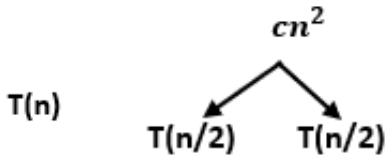
- The function f(n) may represented constant or any polynomial in n.
- T(n) is interpreted as the time required to solve the problem of size n.
- Recurrence of linear search

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

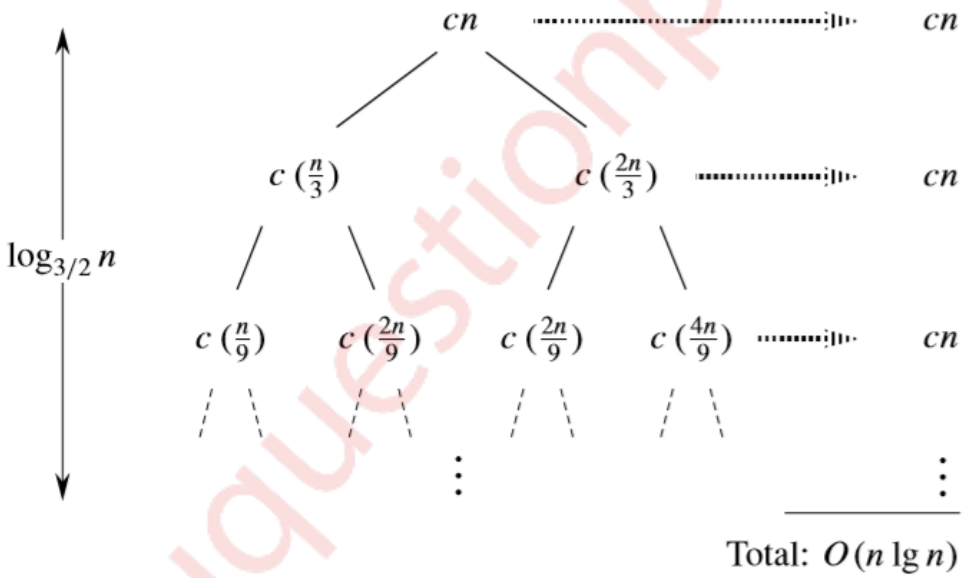
i) Forward substitution ii) Backward substitution

- **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.




**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.




**********

Please check whether you have got the right question paper.

N.B: Subject AOA CSC 402 CBSGS R-12 SE comp SEM IV CBSGS

1. Q.1 is compulsory.
2. Solve any three from Remaining

**Q. 1** Answer **any four**

a) Write an algorithm for finding maximum and minimum number from given set.

b) Write the algorithm and derived the complexity of Binary Search algorithm.

c) Explain masters method with example

d) Write a note on flow shop scheduling

e) Compare divide and conquer, dynamic programming and Backtracking approache u algorithm design.

**Q. 2** a) Write and explain string matching with finite automata with an example

b) Explain how branch and bound strategy can be used in 15 puzzle problem.

**Q. 3** a) What is 0/1 knapsack and fractional knapsack problem.

Solve following using 0/1 knapsack method

| Item (i) | Value (vi) | Weight(wi) |
|---|---|---|
| 1 | 18 | 3 |
| 2 | 25 | 5 |
| 3 | 27 | 4 |
| 4 | 10 | 3 |
| 5 | 15 | 6 |

Knapsack capacity=12.

b) Explain insertion sort and derive its complexity

**Q. 4** a) What is a binary search tree? How to generate optimal binary search tree

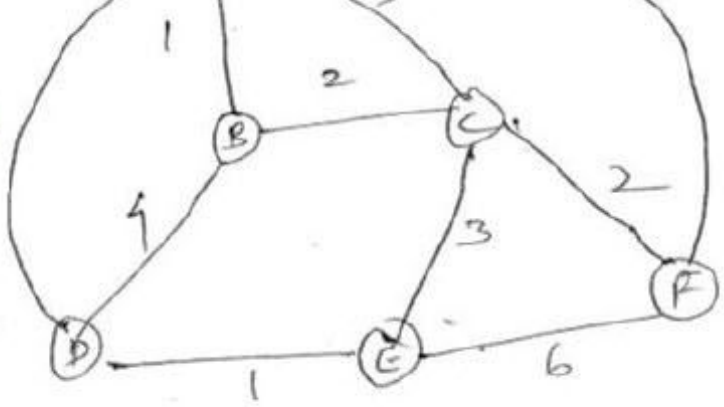
b) What is a longest common subsequence problem? Find LCS for following string X = Y = ABC ABE

**Q. 5** a) Explain Job Sequencing with deadlines.

Let n=4, $(P_1\ P_2\ P_3\ P_4)$ =(100,10,15,27) and $(d_1\ d_2\ d_3\ d_4)$ (2,1,2,1) find feasible solution

b) Explain prims algorithm and find minimum spanning tree for the following graph.

6




Q.6 Write short notes (**any three**):-

a) Problem of multiplying Long Integers
b) Strassen's matrix multiplication
c) Knuth Morris Pratt's Pattern matching
d) Multi stage Graphs

**********

**Analysis of Algorithm**
**(Dec 2019)**

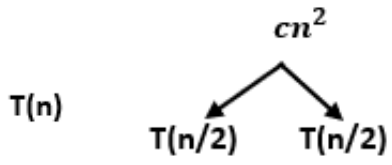
**Q.P. Code - 78093**

**Q 1**

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

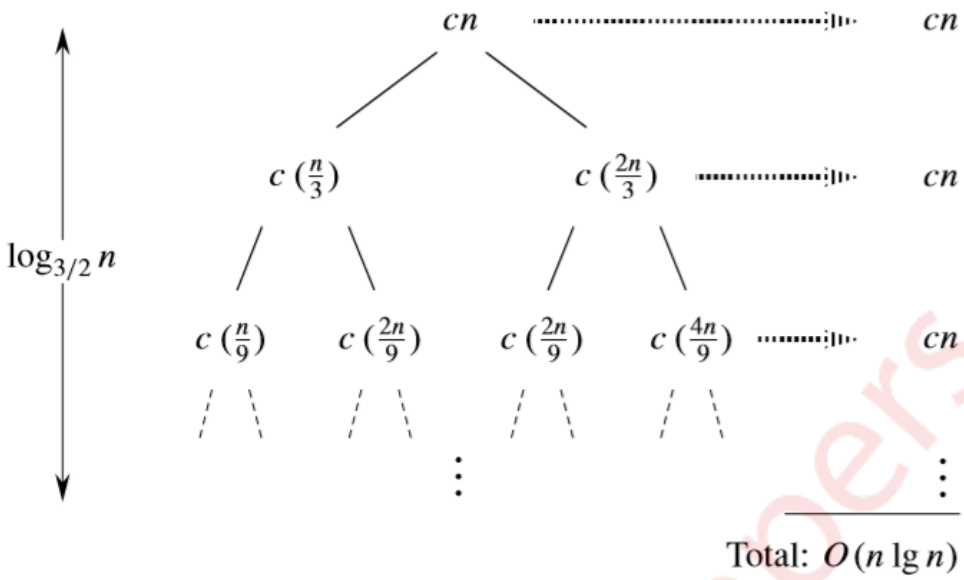
Definition:
Recurrence equation recursively defines a sequence of function with different argument, behavior of recursive algorithm is better represented using recurrence equations.
- Recurrence are normally of the form
T(n) = T(n-1) + f(n), for n > 1
T(n) = 0, for n = 0
- The function f(n) may represented 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**

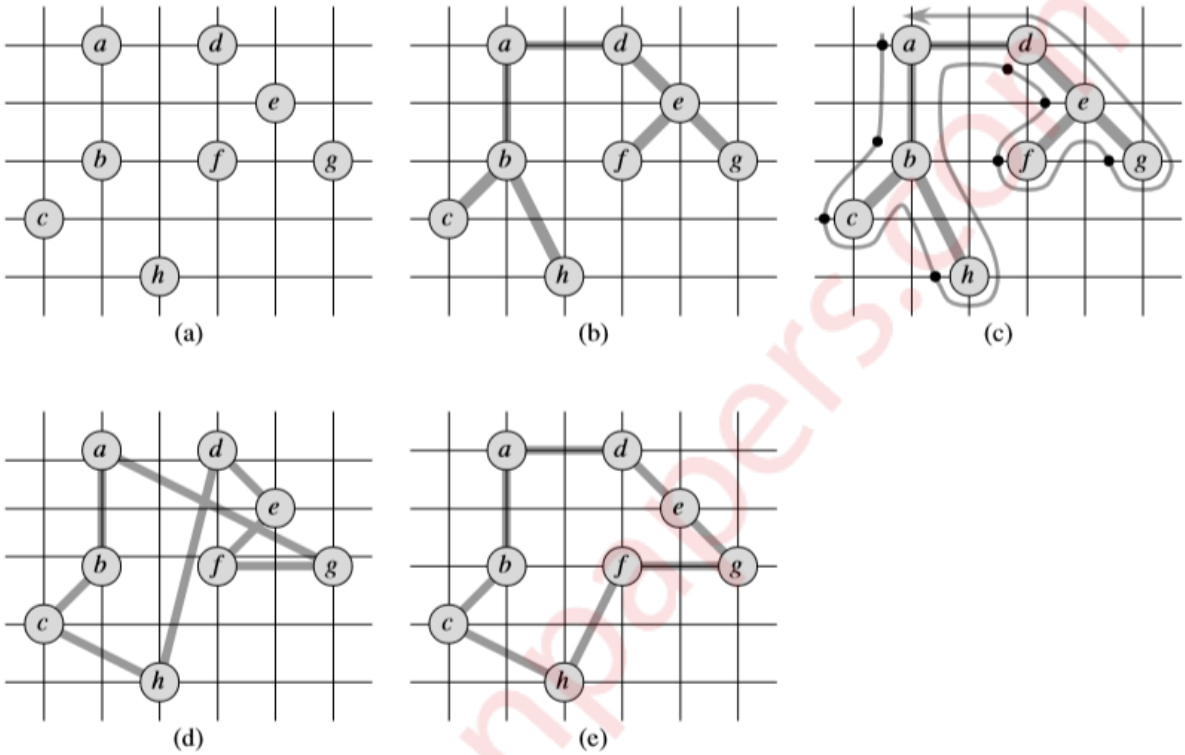
| | **Kruskal's Algorithm** | **Prims Algorithm** |
|---|---|---|
| 1. | This algorithm is for obtaining minimum spanning tree but it is not necessary to choose adjacent vertices of already selected vertices. | This algorithm is for obtaining minimum spanning tree by selecting the adjacent vertices of 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)

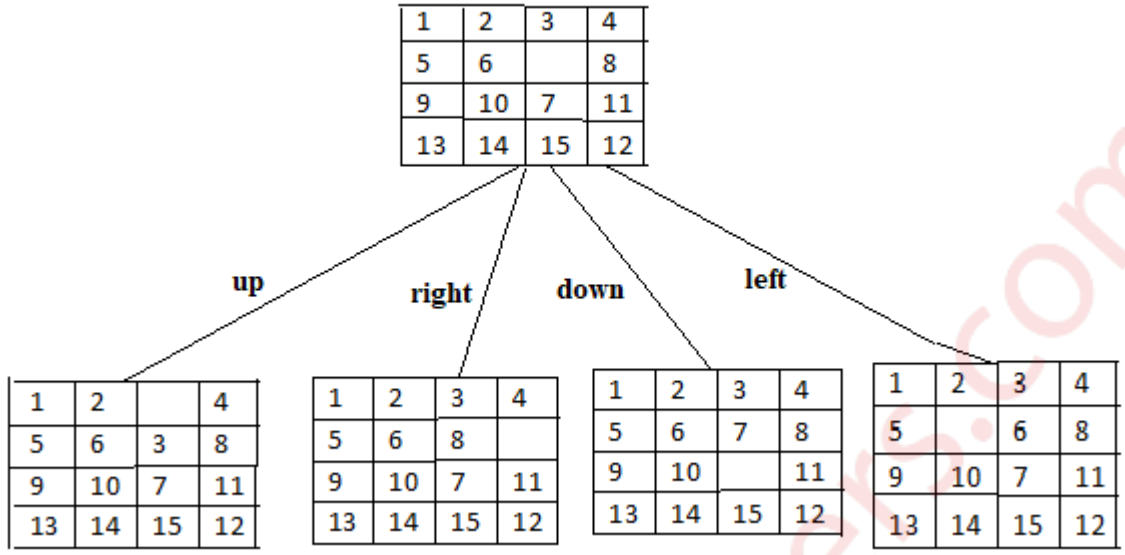



**Fig. Cost f(x) + h(x) after first move**

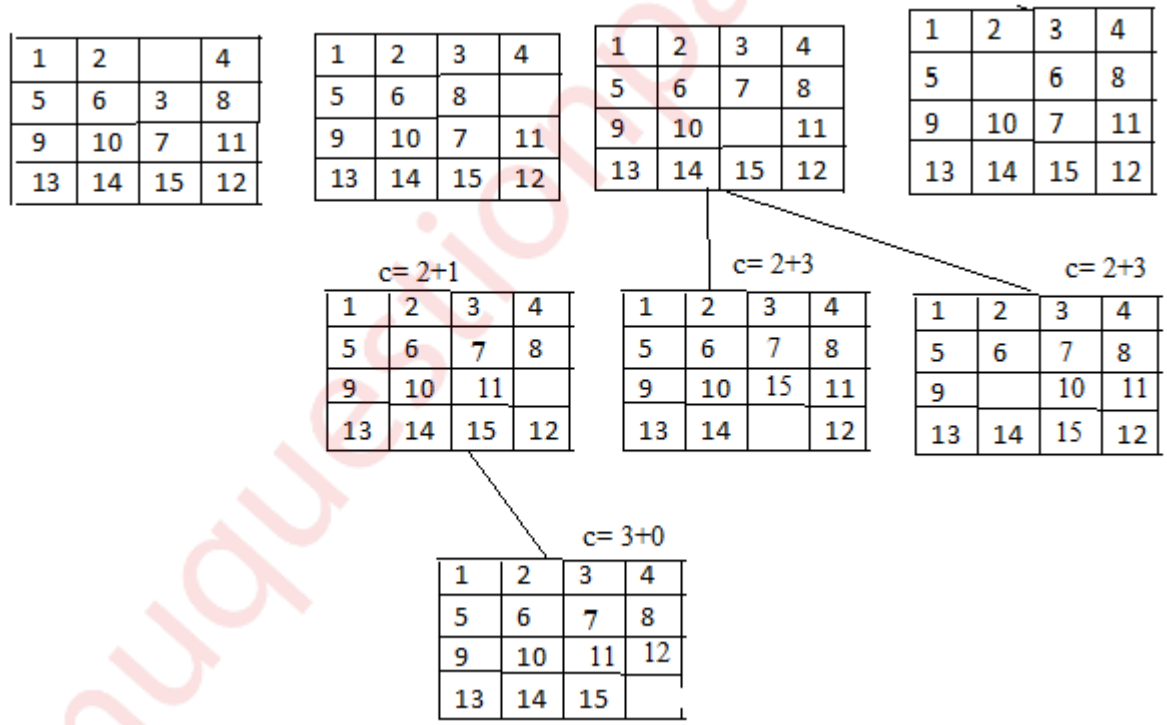
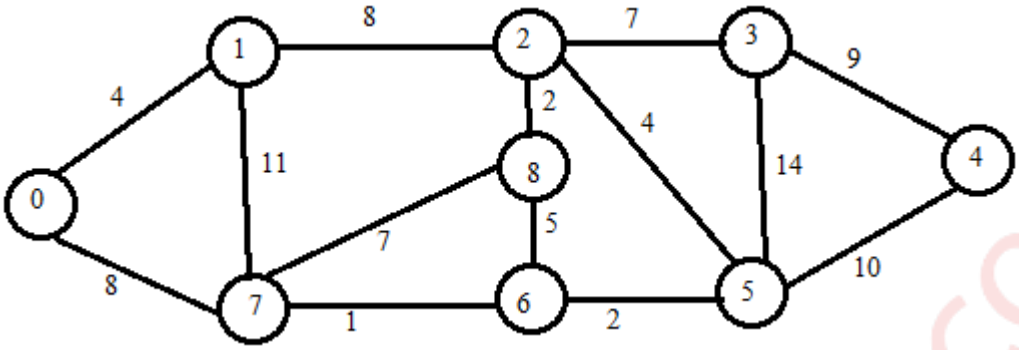



**Fig. Tile positions after subsequent moves**

**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)**

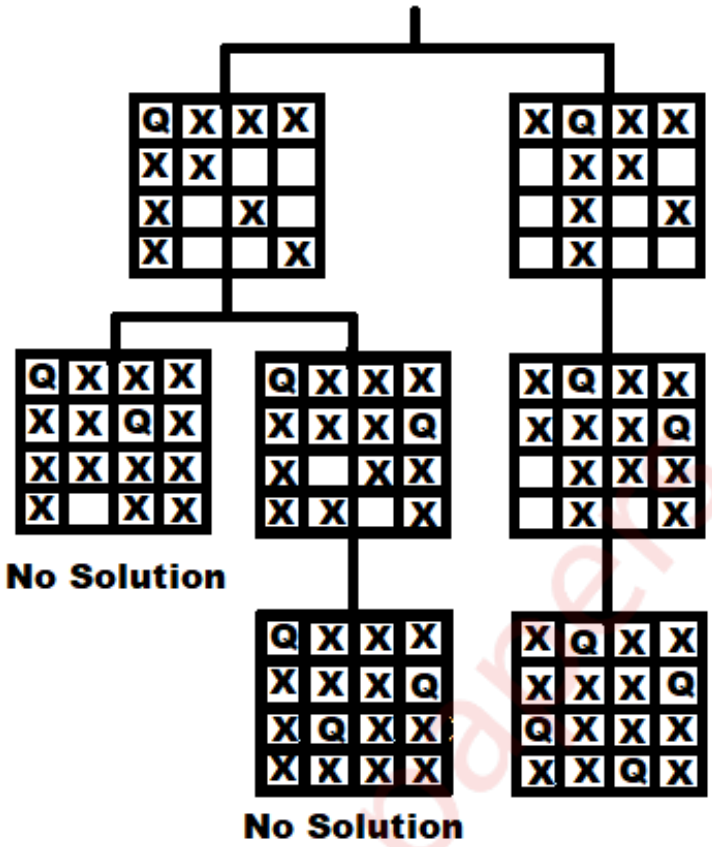
**b) Explain Backtracking with N-queen problem** **10M**

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 $\sum_{i=1}^{n} x_i w_{i \leq M}$

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.

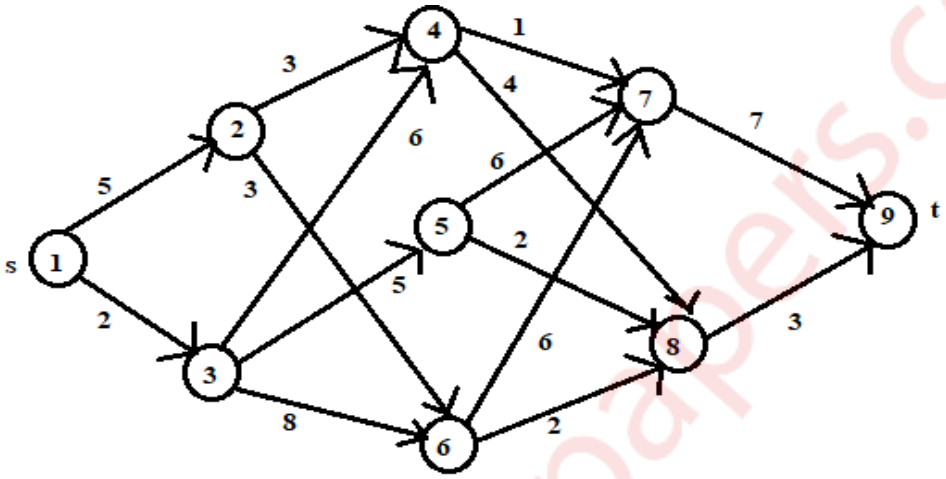
We will discuss two approaches for solving knapsack using dynamic programming.

| **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:



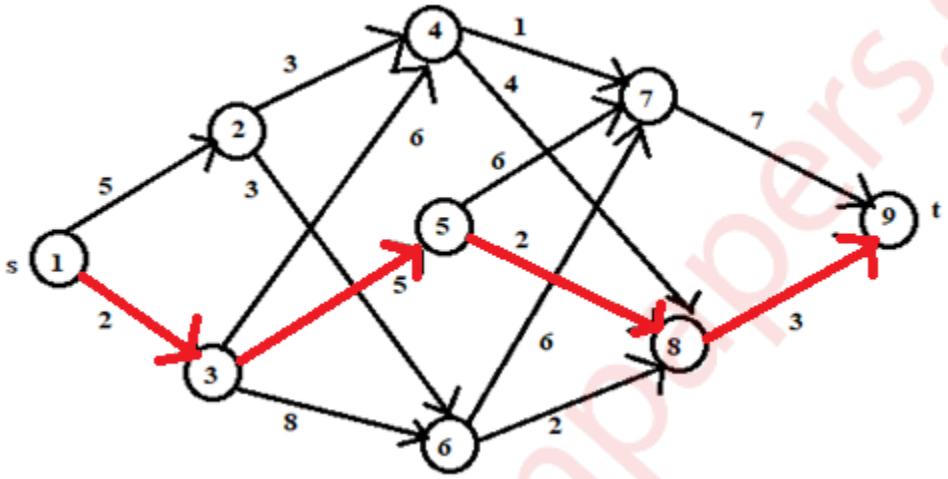

**Stage 1:**
Vertex 1 is connected to 2 and 3
Cost[1] = min{c[1, 2] , c[1, 3] }
= min {5, 2}
= 2
**Stage 2:**
Vertex 2 is connected to 4 and 6
Cost[2] = min {c[2,4], c[2, 6]}
= min{3, 3}
= 3
Vertex 3 is connected to 4, 5 and 6
Cost [3]= min{c[3,4], c[3,5], c[3, 6]}
= min{ 6, 5, 8}
= 5
**Stage 3:**
Vertex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

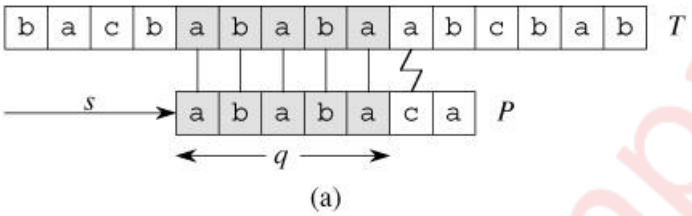
a. This is first linear time algorithm for string matching. It utilizes the concept of naïve approach in some different way. This approach keeps track of matched part of pattern.
b. Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive 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)**

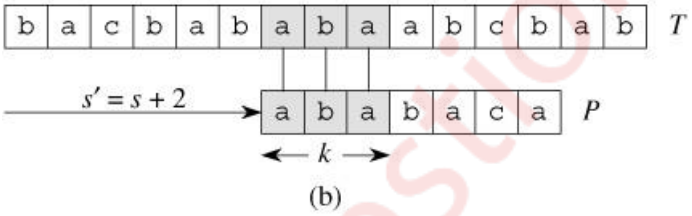
```
n = T.length
m = P.length
∏ = Compute prefix
q  0
for i = 1 to n
while q > 0 and P[q+1] ≠ T[i]
q = ∏ [q]
```

**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 ∏**

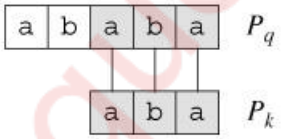
**Example:**



(a)



(b)



(c)

**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.

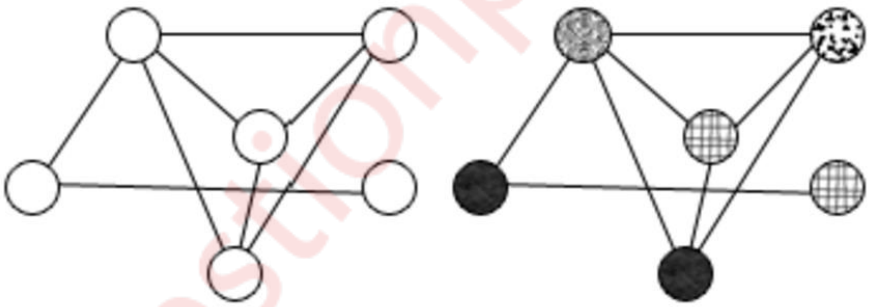
-The problem can be solved simply by assigning a unique color to each vertex,but this solution is not optimal. It may be possible to color the graph with colors less than IvI.

-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.

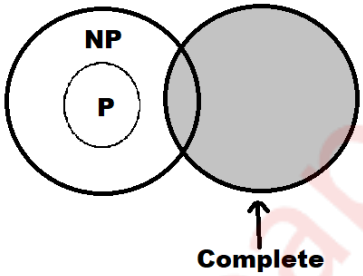
Algorithm – **RABIN_KARP (T, P)**

```
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:**

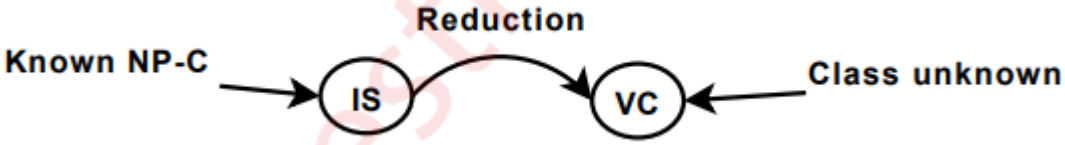
- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.
- Now suppose we have a NP-Complete problem R and it is reducible to Q then Q is at least as hard as R and since R is an NP-hard problem. therefore Q will also be at least NP-hard , it may be NP-complete also.
- 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.

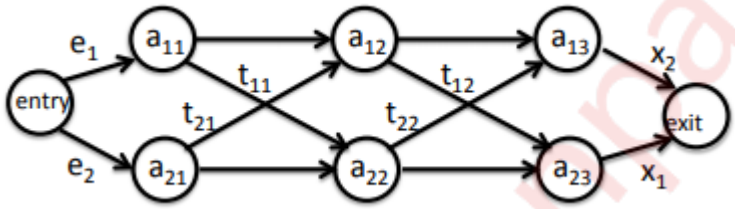
- Manufacturing of large items like car, trucks etc. generally undergoes through multiple stations, where each station is responsible for assembling particular part only. Entire product be ready after it goes through predefined n stations in sequence.

- 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($n^{2.81}$)**

---

**********

Q6. Wri

a) S

b) 8

c) C

d) 1

S.E÷Comp Sem-IV, AOA R-19

QP code: 93328

20 MAY 2022

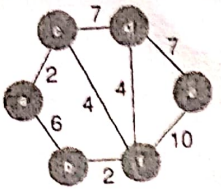
**University of Mumbai**

**Examinations Summer 2022**

Time: 2 hour 30 minutes

Max. Marks: 80

| Q1. (20 Marks) | Choose the correct option for following questions. All the Questions are compulsory and carry equal marks |
|---|---|
| 1. | _____ is the class of decision problems that can be solved by non-deterministic polynomial algorithms. |
| Option A: | NP |
| Option B: | P |
| Option C: | Hard |
| Option D: | Complete |
| | |
| 2. | Following data structure is used to implement LIFO Branch and Bound Strategy |
| Option A: | Priority Queue |
| Option B: | array |
| Option C: | stack |
| Option D: | Linked list |
| | |
| 3. | For the given elements 6 4 11 17 2 24 14 using quick sort, what is the sequence after first phase, assuming the pivot as the first element? |
| Option A: | 2 4 6 17 11 24 14 |
| Option B: | 2 4 6 11 17 14 24 |
| Option C: | 4 2 6 17 11 24 14 |
| Option D: | 2 4 6 11 17 24 14 |
| | |
| 4. | Which of the following is correct for branch and bound technique?<br>i. It is BFS generation of problem states<br>ii. It is DFS generation of problem states<br>iii. It is D-search. |
| Option A: | Only i |
| Option B: | Only ii |
| Option C: | Only ii and iii |
| Option D: | Only i, and iii |
| 5. | Consider the given graph. |

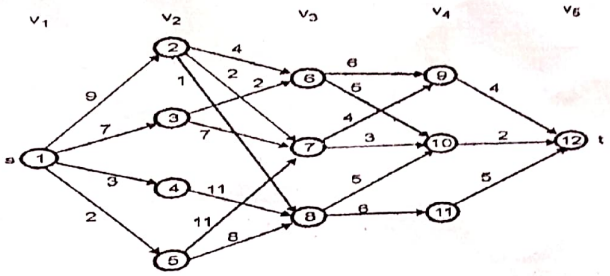
| | |
|---|---|
| |  <br>What is the weight of the minimum spanning tree using the Kruskal's algorithm? |
| Option A: | 24 |
| Option B: | 23 |
| Option C: | 15 |
| Option D: | 19 |
| | |
| 6. | Bellman Ford algorithm is used to find out single source shortest path for negative edge weights. Bellman Ford algorithm uses which of the following strategy? |
| Option A: | Greedy method |
| Option B: | Dynamic Programming |
| Option C: | Backtracking |
| Option D: | Divide and Conquer |
| | |
| 7. | The optimal solution for 4-queen problem is |
| Option A: | (2,3,1,4) |
| Option B: | (1,3,2,4) |
| Option C: | (3,1,2,4) |
| Option D: | (2,4,1,3) |
| | |
| 8. | Consider the following code snippet:<br>Bounding function(k,i) {<br>for(j=1 to k-1)<br>{ if ((x[j]==i) or (Abs(x[j]-i) ==abs(j-k))) return false;<br>} return true }<br>The above code represents the bounding function for which of the following algorithm? |
| Option A: | Subset sum problem using backtracking |
| Option B: | n-queens using backtracking |
| Option C: | Graph coloring using backtracking |
| Option D: | Subset sum using branch and bound |
| | |
| 9. | What do you mean by chromatic number? |
| Option A: | The minimum number of colors needed to color all the vertices optimally in a Graph |

| | |
|---|---|
| | Coloring problem |
| Option B: | The maximum number of colors needed to color all the vertices optimally in a Graph Coloring problem |
| Option C: | The number of colors using which the edges of graph have been colored in a Graph Coloring Problem |
| Option D: | The individual colors with which we color the vertices of a Graph in a Graph Coloring Problem |
| | |
| 10. | Which string matching algorithm uses a Prefix Table? |
| Option A: | Naïve String Matching Algorithm |
| Option B: | Boyer Moore String Matching Algorithm |
| Option C: | Knuth Morris Pratt Algorithm |
| Option D: | Rabin Karp Algorithm |

| **Q2.** **(20 Marks)** | **Solve any Four out of Six** **05 marks each** |
|---|---|
| A | Write and Explain binary search algorithm. |
| B | Write a short note on job sequencing with deadline |
| C | Determine the LCS of the following sequences:<br>X: {A, B, C, B, D, A, B}<br>Y: {B, D, C, A, B, A} |
| D | Solve the sum of subsets problem for the following: n=4, m=15, w={3,5,6,7} |
| E | Give the algorithm for the N-Queen's problem and give any two solutions to the 8-Queen's problem |
| F | Explain and apply Naïve string matching on following strings<br>String1: COMPANION<br>String2: PANI |

| **Q3.** **(20 Marks)** | **Solve any Two Questions out of Three** **10 marks each** |
|---|---|
| A | Write algorithm for greedy knapsack and Obtain the solution to following knapsack problem where n=7,m=15 (p1,p2.....p7) = (10,5,15,7,6,18,3), (w1,w2,.....,w7) = (2,3,5,7,1,4,1). |
| B | Explain Dijkstra's Single source shortest path algorithm. Explain how it is different from Bellman Ford algorithm. Explain 15-puzzle problem using LC search technique. |
| C | Rewrite and Compare Rabin Karp and Knuth Morris Pratt Algorithms Give the pseudo code for the KMP String Matching Algorithm. |

| **Q4.** **(20 Marks)** | **Solve any Two Questions out of Three** **10 marks each** |
|---|---|
| A | Write algorithm for quick sort and sort the following elements [40,11,4,72,17,2,49] |
| B | Write multistage graph algorithm and solve following example. |

| | |
|---|---|
| |  |
| C | Write algorithm for 0/1 knapsack problem using dynamic programming .Also solve the following example.<br>N=4, M=21 (p1,p2,p3,p4)=(2,5,8,1),(w1,w2,w3,w4)=(10,15,6,9) |