PAGE NO.
DATE / /

Q7 What is greedy algorithm with example.

Ans:

- The greedy method is one of the stategies like divide and conquer algorithms used to solve problems
- This method used solving optimization problems that demand either maximum or minimum results.

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment

- It doesn't worry whether the current best result will bring the overall optimal result.
- it's works in a top-down approach
- This algorithm may not produce the best result for all the problems. it's because it always goes for the local best choices to produce the global best result.

Characteristic of greedy algorithms

- Greedy choice property : A globally optimal solution can be arrived at by choosing a locally optimal choice
- optimal substructure : A problems has an optimal substructure if an optimal soln to the problem contains optimal soln to its ~~subsets~~ subproblems.

Example: Coin change Problem (Greedy Approach)

Problem: Suppose you need to make ₹93 using the least number of Indian currency coins

₹1, 2, 5, 10, 20, 50

Greedy Approach:

- pick the largest denomination possible without exceeding ₹93
- subtract this value from ₹93
- Repeat until the amount becomes 0

Steps:

1. take ₹50 → Remaining 43
2. Take ₹20 → Remaining 23
3. take ₹20 → Remaining 3
4. Take ₹2 → Remaining 1
5. take ₹1 → Remaining 0

Coined used: 50, 20, 20, 2, 1 (total 5 coins)

Q) What is Dijkstra's Algorithm with example

Ans:

~~Dijkstra~~

- Dijkstra algorithm is used to find the shortest path from a single source vertex to all other vertices in a weighted graph
- It is greedy algorithm because it always picks the shortest known path at each step.

Steps of Dijkstra algorithm:

(1) initialize: Set the distance of the source node 0 and all other nodes to ∞ infinity

(2) choose the minimum distances Node:

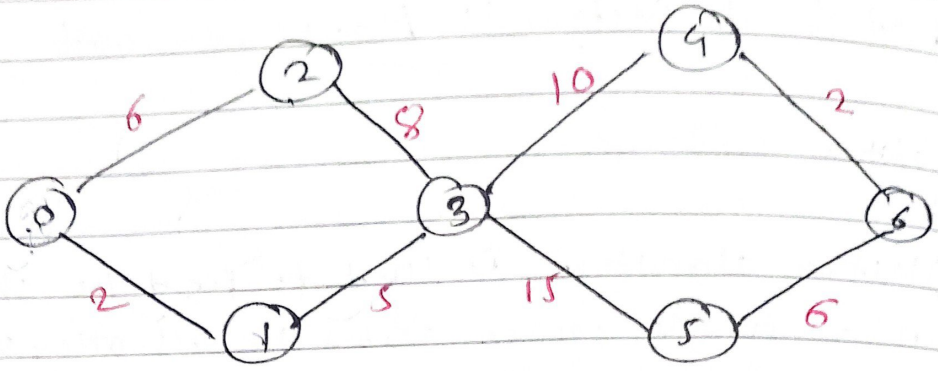
Pick the node with smallest known distance that hasn't been processed yet.

(3) update distances

(4) Repeat until all nodes have been processed.

PAGE NO.
DATE / /

example :



0 → 1 = .
0 → 2
0 → 3
0 → 4
0 → 5
0 → 6

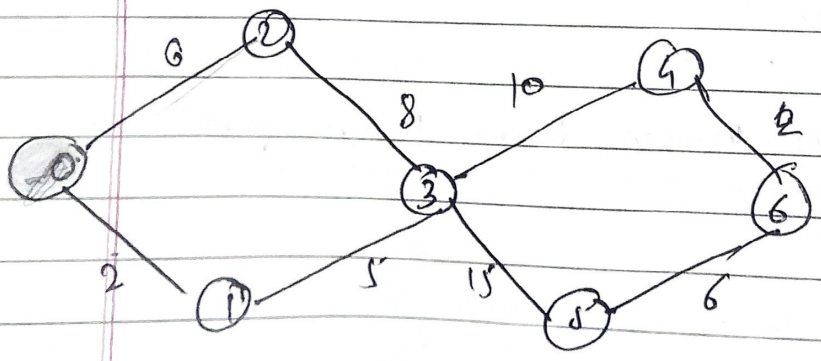
The distance from source node to itself is zero

Source to other node infinity (∞)

Step ① Start from Node 0 and mark Node as visited.

Unvisited Nodes

{ 0, 1, 2, 3, 4, 5, 6 }



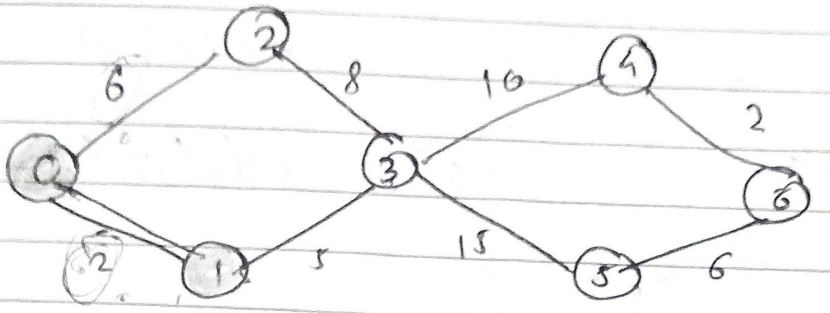
distance

0 : 0 ✓
1 : ∞ 2
2 : ∞ 6
3 : ∞ 7
4 : ∞ 17
5 : ∞ 22
6 : ∞ 17

check for the adjacent Node, we have two choice either Node ① or ② choose minimum distance.

for Node ①

Step ②

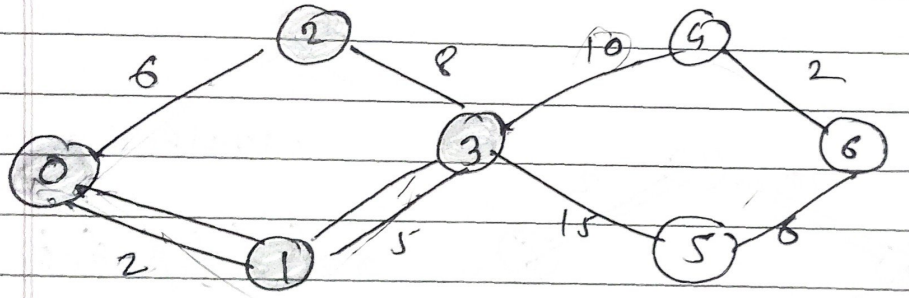


Unvisited Nodes

{0, 1, 2, 3, 4, 5, 6}

distance : 0 → 1 = 2

Step ③ Then move forward and checks Adjacent ~~dist~~ Nodes which is ③



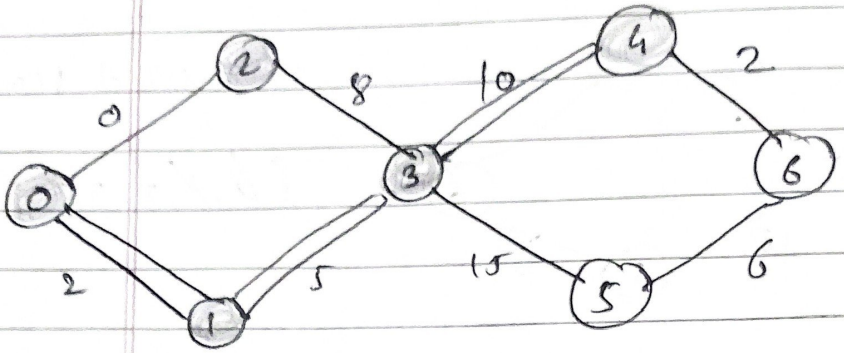
Unvisited Nodes

{0, 1, 2, 3, 4, 5, 6}

distance : 0 → 1 → 3 = 2 + 5 = 7

* Since we can't go 0 → 2 → 3 = 6 + 8 = 14

Step ④ Again we have two choice for adjacent Nodes so choose the minimum Node with minimum distance which is Node ④



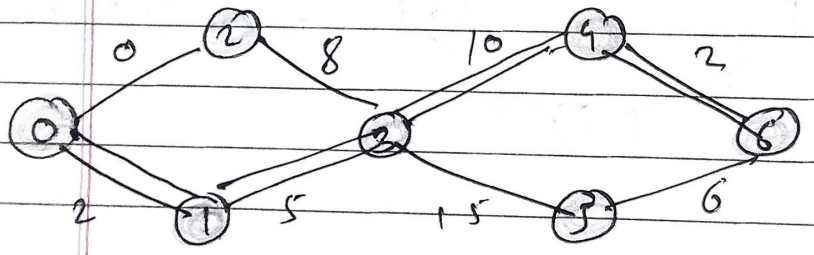
unvisited

{ 0, 1, 2, 3, 4, 5, 6 }

distance: $0 \to 1 \to 3 \to 4 = 2 + 5 + 10$

$= 17$

Step (5) Again move forward and check for Adjacent Node which is (6)



unvisited Node

{ 0, 1, 2, 3, 4, 5, 6 }

distance: $0 \to 1 \to 3 \to 4 \to 6 = 2 + 5 + 10 + 2$

$= 19$

So the shortest distance from 0 to 6 is 19

which is optimal

Q) Diff b/w Kruskal & Prim's algorithm

→

NO:	Prim's algorithm	Kruskal's Algorithm
(1)	It first check for the feasibility and then for optimality	It first checks for the optimality, and then for the feasibility.
(2)	No sorting of edges of a graph is required.	It arrange all edges of a graph in ascending order.
(3)	At every ~~edge~~ stage, one and only one connected component is formed.	At every stage, one or more than one connected components are formed.
(4)	generally used data structure is a weighted adjacency matrix	Generally used data structure is priority queue.
(5)	Starts from an arbitary vertex	Start with all vertices as separate trees (forest)
(6)	dense graph	~~sph~~. sparse graphs
(7)	difficult to parallerize	easier to parallelize.
(8)	TC: O(E log V)	TC: O (E log E)

* A min spaning tree (MST) is a spanning tree of weighted graph that has the minimum possible sum of edge weight connecting all vertices without forming any cycle.

PAGE NO.
DATE / /

Q) Define minimum spanning tree

Ans:

- A spanning tree of connected graph is it's connected a cyclic subgraph (tree) that contains all the vertices of a graph.

- MST of weighted connected graph is a spanning tree is defined as the sum of the weights on all edges

- The MST problems is a problem of finding mst for given weighted connected graph.

* Algorithms to find MST
  - Kruskal's
  - Prim's

: Both are greedy approach.

Q) Why AOA is Imp

Ans:

Analysis of Algorithms (AOA) is essential in computer science because it helps in understanding the efficiency of different algorithms in terms of time and space.

It allows developer to choose the best algorithm for a given problem, ensuring optimal performance

- Key Reason why AOA is important

- Performance optimization
- scalibility
- comparision of Algorithms
- Cost Reduction
- Theoretical Understanding
- Real world application
- Competitive programming & interviews

Understanding AOA ensures that software run faster, consumes fewer resources and scale well for large inputs.
It is essential skill for every programmer and software engineers

IMP any ① coding ask

PAGE No.
DATE / /

★ Sorting algorithms:

(1): **Merge Sort:** A divide and conquer sorting algorithm that splits an array into two halves, recursively sort each half, and then merges them in sorted order.

Algorithm:

1. if the array has one or zero elements, return
2. divide the array into two halves
3. Recursively sort both halve using merge sort
4. Merge the sorted halves back into a single Sorted Array

TC → O(n log n) — Best
SC → O(n) — Auxiliary Space

give example of each algo!

(2) quick sort : A divide and conquer algorithm that selects a pivot, partitions the array around the pivot and recursively sorts both partitions.

Algorithm :

1. Select the pivot element from the array (highest one)
2. Partition the array such that element smaller than pivot are on the left and larger than ones on the right.
3. Recursively apply quick sort to the left and right sub arrays.

TC → O(n log n) → Best

SC → O(n log n) → due to recursion

Comparision - based

↑

(3) Selection Sort : A simple comparision - based sorting algorithm that repeatedly selects the smallest elements from the unsorted part of the array and swaps it with first unsorted element.

Algorithm:

1. Find the smallest elements in the array
2. Swap it with the first element
3. Repeat the process for the remaining unsorted part of the array.

TC → $O(n^2)$

SC → $O(1)$

(a) Insertion sort: A simple and efficient sorting method for small datasets where element are picked one by one and place in the correct position within the sorted part of the array.

Algorithm:

1. Start from the second elements, assume the first one is sorted
2. compare the element with previous ones and shift elements if needed
3. Insert the elements at the correct position
4. Repeat until the entire array is sorted.

T(→ O(n)

S(→ O(1)

PAGE No.
DATE / /

Q) What is Recurrence? Various Method to solve recurrence

Ans:

A recurrence Relation is a mathematical expression that defines a sequences in terms of it's previous terms.

in context of algorithmic analysis, it is often used to model the time complexity of recursive algorithm.

The general form:

$$an = (F)(an-1, an-2 \text{ ---- } an-k)$$

where F is function, that defines the relationship between the current term and previous terms

The recurrence relation display significant roles in analysing and optimising the complexity of algorithms.

- Three ways to solving Recurrence:

① Substitution Method

② Recurrence Tree Method

③ Master Method

(1) Substitution method: (guess and verify)

- guess the soln form and use mathematical induction to prove it.

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

assume $T(n) = O(n \log n)$

(2) Recursion Tree method:

- expand the recurrence into a tree to identify patterns and derive the complexity

eg: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

$$T(n) = 2T(n/2) + n$$

$$= 4T(n/4) + 2n$$

$$= 8T(n/8) + 3n$$

which leads to $O(n \log n)$

(3) Master theorem:

- used for divide and conquer recurrence of the form

$$T(n) = aT(n/b) + f(n)$$

$f(n) = n^d$ — recursive work

a ⇒ no. of sub problems

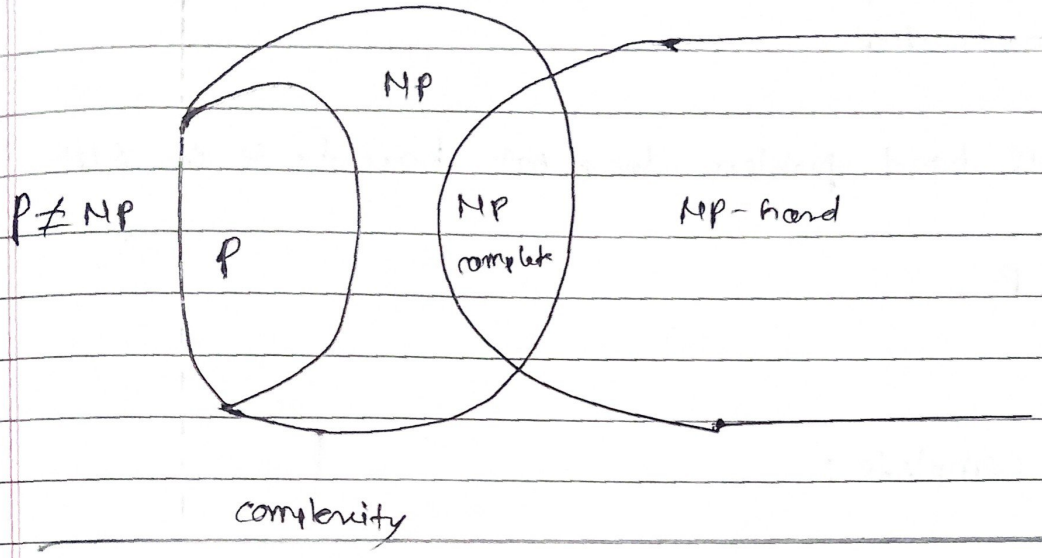
b ⇒ factor by the problem which size is divided

Sem-4

PAGE No.
DATE / /

Q) describe the relationship along P, NP, NP-hard, NP-complete

→



(1) P (polynomial time) :

The class of decision problems that can be solved by deterministic Turing machine in polynomial time.

eg: BS, Sorting

(2) NP (Non-deterministic Polynomial Time):

This is a class of decision problem where a given solution can be verified in polynomial time, even if finding the soln might take longer.

eg: sum of subset problem

PAGE No.
DATE / /

(3) NP-Hard:

- A class of problem at least as hard as the hardest problem in NP
- An NP hard problem does not have to be in NP

eg: TSP

(4) NP-complete:

A problem is NP-complete if it's satisfied two condn:

(i) it belong to NP
(ii) it is hard as every problem in NP

eg: SAT

Q) What is Asymptotic analysis and define big o, big omega and big theta notation.

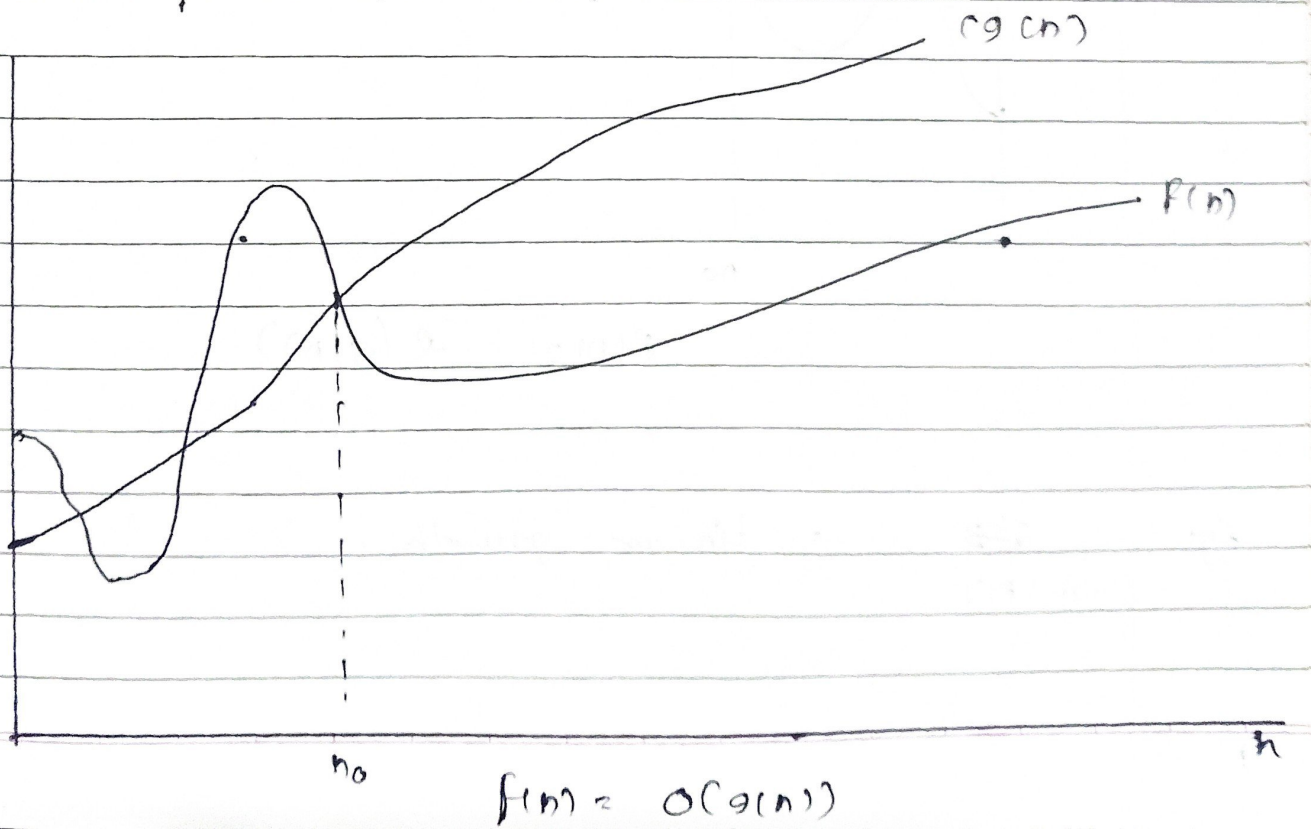
Ans:

Asymptotic Analysis is a method of describing the efficiency of an algorithm as the input size (n) becomes very large.

it helps us understand the time or space complexity of an algorithm without worrying about machine specific constant or small inputs.

(1) Big - O Notation (O): Worst case:

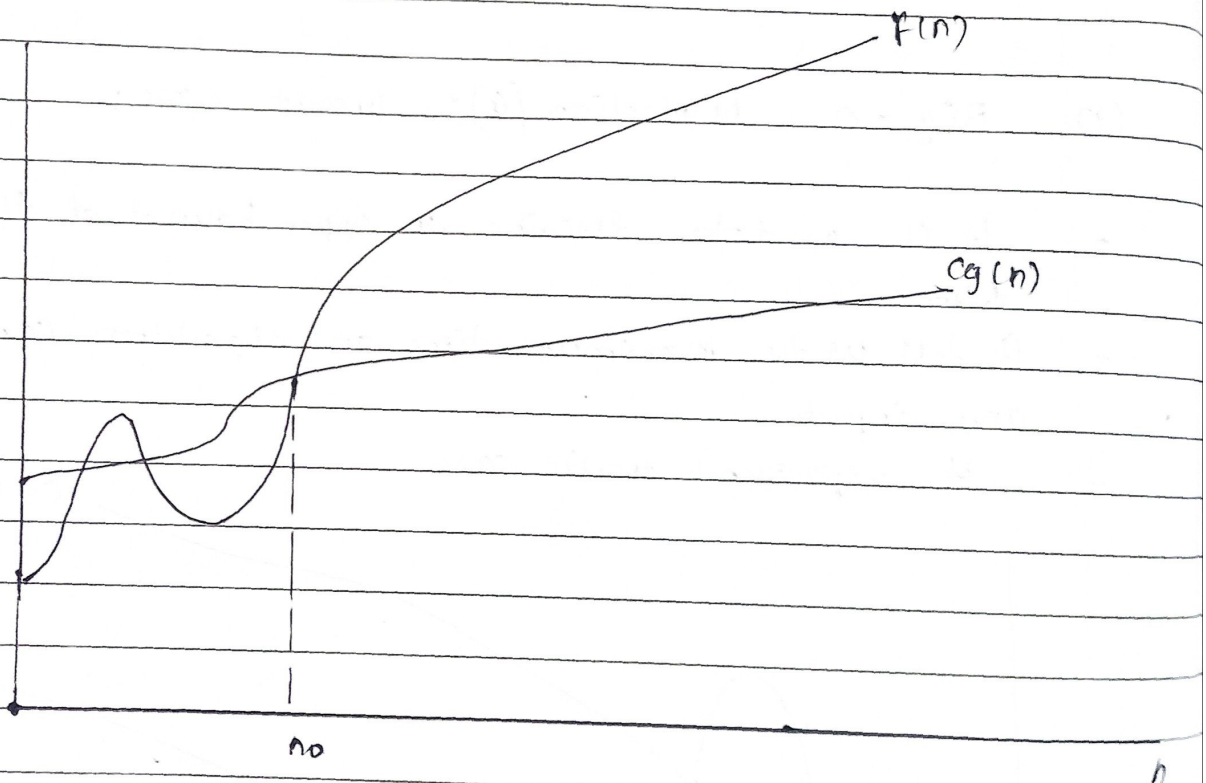
- it is used to describe an upper bound of the running time.
- it tell us the maximum time an algorithm can take for any input
- it represent worst case.



eg: $O(n) \rightarrow$ Linear
$O(1) \rightarrow$ constant
$O(n^2) \rightarrow$ quadratic
} TC

(2) Big - Omega Notation (Ω): Best case

- it is used to describe the lower bound of the running time
- it tells minimum time an algorithm will take.
- it represent the best case scenario.

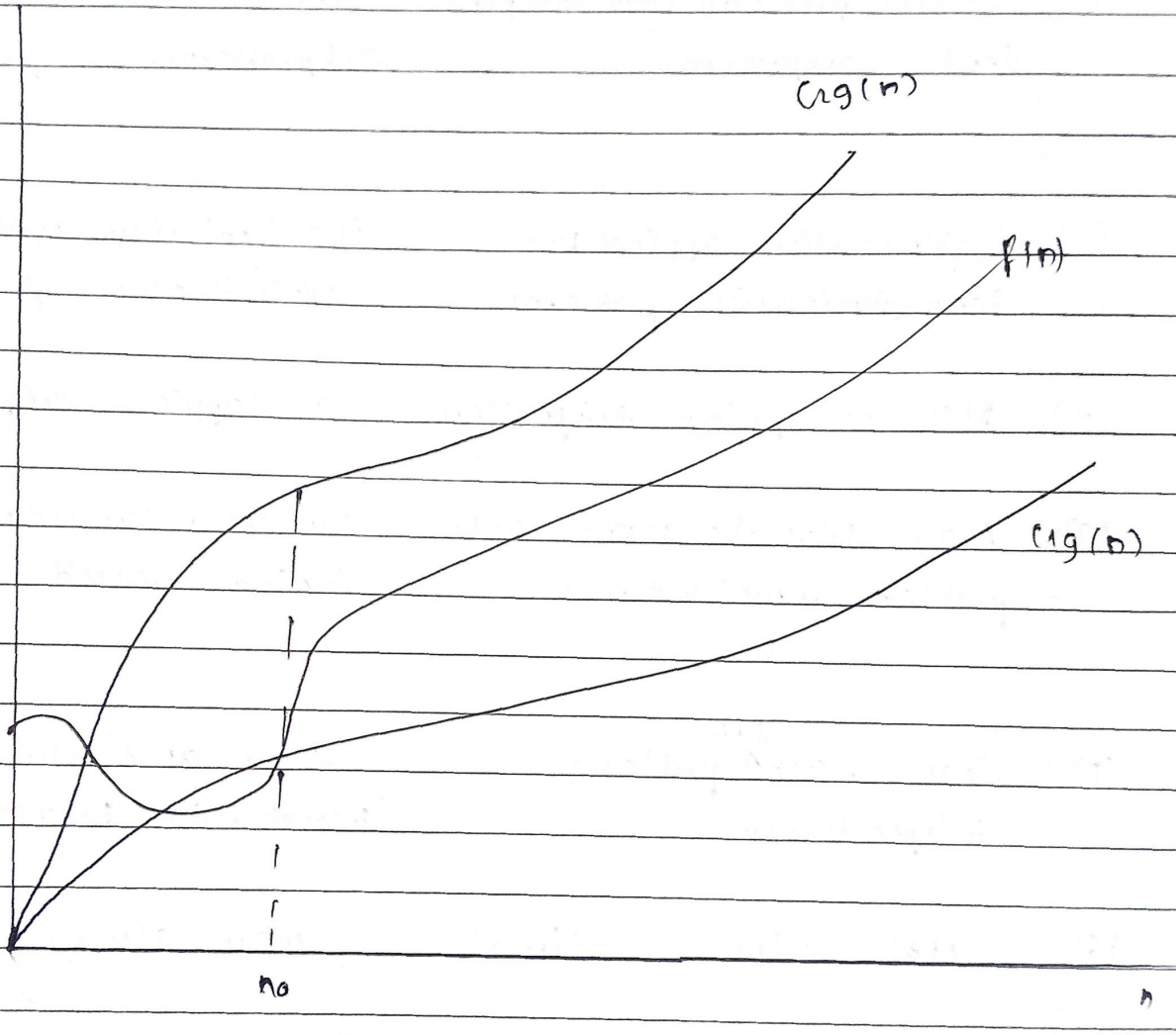


$f(n) = \Omega(g(n))$

eg: $\Omega(n) \rightarrow$ Linear growth

(3) Theta Notation (Θ) : Average case

- it is used to describe exact / tight bound of the running time
- it gives both upper and lower bound
- it represent average case scenario.



$$f(n) = \Theta(g(n))$$

eg: $\Theta(n \log n)$

PAGE NO.
DATE / /

Q) diff b/w divide & conquer & Dynamic programming

→

Sr. NO	Divide and conquer	Dynamic programming
(1)	divides problem into independent subproblems	divides into overlapping subproblems.
(2)	It generally applied for non-optimization problems	it is typically applied for optimization problem.
(3)	NO, overlapping subproblem	overlapping subproblems
(4)	May solve the same sub-problem multiple times	Avoids recomputation by storing result.
(5)	Solves each sub problems independently	Stores and reuses solution (memorization / tabulation)
(6)	Not always optimal	may always optimal
(7)	eg: Merge sort, quick sort, Binary search	eg: Fibonacci series, 0/1 knapsack

q) greedy knapsack and 0/1 knapsack

→

Sr. No	Greedy knapsack (Fractional)	0/1 knapsack. (DP)
(1)	item can be broken into fractions	item must be taken completely or not at all
(2)	greedy approach use	dynamic programming use
(3)	always ~~not~~ gives optimal solution	Gives optimal solution
(4)	Require less memory	Require more memory due to DP table.
(5)	Strategy : pick item with highest value / weight ratio first	Strategy : explore all possible combination
(6)	TC : O(n log n) due to sorting	O(n * W) (when W is capacity)
(7)	SC : O(1) or O(n)	O(n * W)
(8)	Use case : when fractions are allowed eg : gold	when only whole item can be chosen. eg : laptop

PAGE NO.
DATE / /

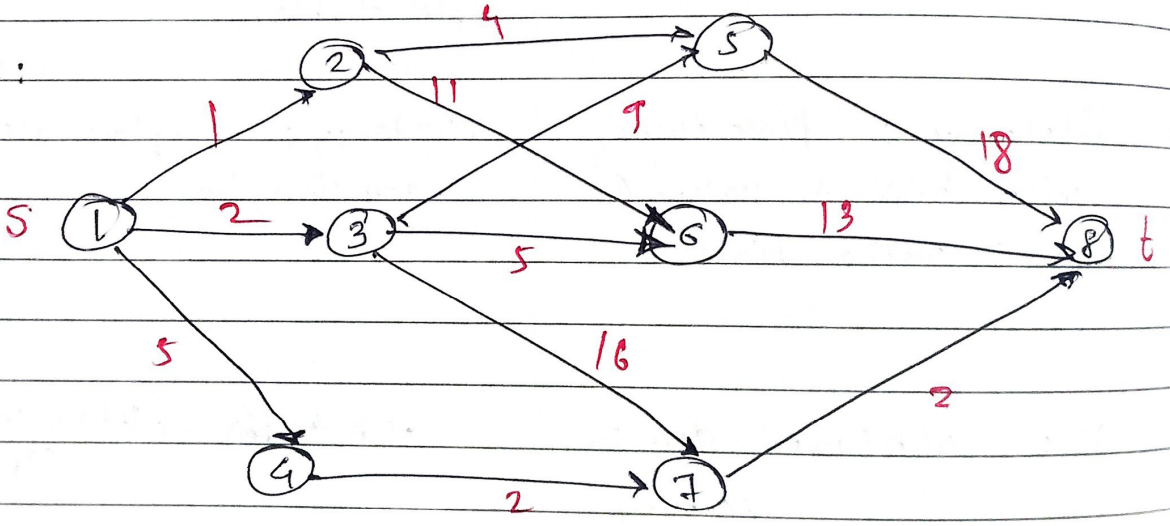
Q) Explain Multi-stage graph with example.

→

A multistage graph is a directed, weighted graph that is divided into multiple stages where,

- Nodes are arranged in stages from start to end
- Edges go only from one stage to next stage (never backward)
- The goal is usually to find the shortest (or longest) path from the source node (in stage 1) to destination node (in the last stage)

Example:



$$f(cost(i,p) = \min \{ c(p,q) + f(cost(i+1,q) \}$$

f(cost(i, p) gives the cost of the shortest path from node p in stage i to the sink node

PAGE No.
DATE / /

(1) Write an abstract algorithm for a greedy design method.

→

Algorithm Greedy-Subset (p, n)

input: p[1:n] is a set of n inputs of a given problem instances.

output:

```
{
    S := ϕ
    for (i:= 1 ; i ≤ n ; i++) {
        q := Select (p);
        if ( Feasible (S, q) );
        S := Union ( S, q);
    }
    return S;
}
```

9) Write an algorithm to find min & max values using divide & conquer and also derive its time complexity. & S.C

→

To find out minimum and maximum number of an array by performing lesser operation the divide and conquer strategy is used.

Approach: The divide and conquer approach works by splitting the array into two halves, recursively finding the minimum and maximum in each half and then merging the results.

Algorithm:

```
MinMax (A, low, high):
  if (low == high):                    // only one element
      return (A[low], A[low])

  else if high == low + 1:             // two elements
      if A[low] < A[high]:
          return (A[low], A[high])
      else:
          return (A[high], A[low])

  else:
      mid = (low + high)/2
  (min1, max1) = MinMax (A, low, mid)
  (min2, max2) = MinMax (A, mid+1, high)

  min_final = min(min1, min2);
  max_final = max(max1, max2);

  return (min_final, max_final)
```

PAGE No.
DATE / /

derive TC :

Let T(n) be the no. of comparision of an array size n

$T(n) = T(n/2) + T(n/2) + 2$

$= 2T(n/2) + 2$

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

Worst / Avg case = $O(n)$

Best Case = $O(1)$

Q) Write an algorithm and derive it's TC & SC of binary search.

→

Binary search: Binary search is an efficient searching algorithm used to find the position of a target element in a sorted array by repeatedly dividing the search interval in half.

Algorithm: BinarySearch (A, key) :

```
low = 0
high = length(A) - 1

while low <= high
    mid = (low+high)/2

    if (A[mid] == key)
        return mid

    else if (key < A[mid])
        high = mid - 1

    else
        low = mid + 1

return -1          // element not found.
```

PAGE No.
DATE / /

* TC: Let n be the no of elements

At each step, the array is divided into half
So no. of comparisions = $\log_2(n)$

Best case: O(1)
Worst/Avg: O(logn)

* SC: O(1) → iterative
O(logn) → Recursive.

Q) Write Kruskal's algorithm for finding a minimum spanning tree. explain it's working with an example compute T

→

Kruskal algorithm is a classic algorithm used in graph theory to find MST of a connected, direct graph.

The MST is subset of the edges that connect all the vertices without any cycles and with the minimum possible total edge weight.

★ Steps of Kruskal's algorithm:

① Sort all the edges

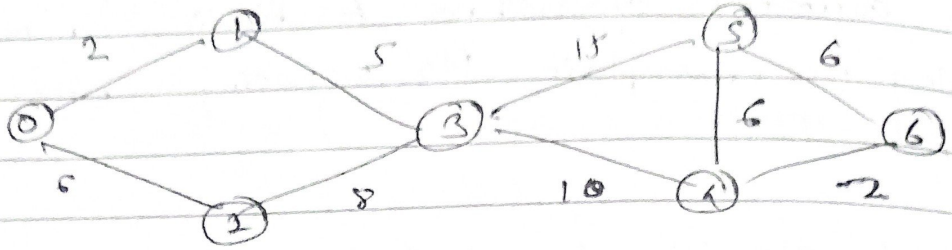
② Initialize subset

③ Iterate

④ sorted edges

⑤ repeat until MST is compute.

eg:



0 → 1 = 2

0 → 2 = 6

1 → 3 = 5

2 → 3 = 8

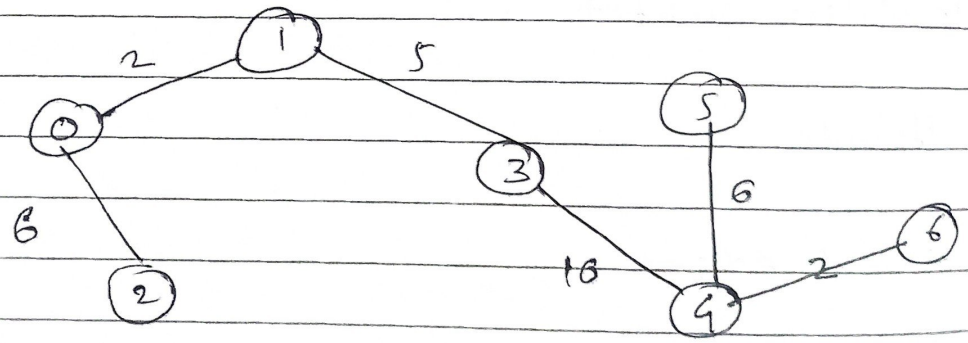
3 → 5 = 15

3 → 4 = 10

5 → 4 = 6

5 → 6 = 6

4 → 6 = 2



6 + 2 + 5 + 10 + 6 + 2

= 30

PAGE NO.
DATE / /

* TC:

V = No. of vertices

E = no. of edges

$O(E \log E) \simeq O(E \log V)$

* SC: $O(V)$

Q) Write a algorithm to solve N queen problem using backtracking.

Ans:

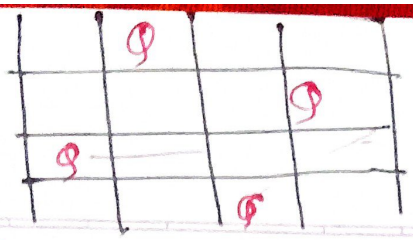
Place N queens into N x N chessboard such that no two queens attack each other (i.e no same row, column or diagonal)

Algorithm: Solve NQueen (board, row, N):

```
if row == N:
    print board / store solution
    return

for col = 0 to N-1:
    if isSafe (board, row, col, N):

        board[row][col] = 1          //place queen
        Solve NQueen (board, row+1, N)
        board[row][col] = 0          //backpack
```

isSafe (board, row, col, N):

```
//check vertical column
for i = 0 to row-1:
    if board[i][col] == 1
        return false

//check left diagonal

for i = row-1, j = col-1 to i >= 0 && j >= 0
    if board[i][j] == 1
        return false
```



//check right diagonal

for i = row-1, j = col+1 to i >= 0 && j < N

if board[i][j] == 1

return false

return true

★ Working

- Place a queen row by row
- for each row, try all column and only place the queen if it's safe
- if all queen are placed (row == N) You've found a valid solution
- Backtrack if no valid column is found for a row

TC = O(N)

SC = O(N²)

PAGE NO.
DATE : / /

Q) Explain travelling salesman problem using Branch & Bound

Ans:

The travelling salesman problem (TSP) is a classic optimization problem in computer science and operation research task,

A sales man must visit a set of cities exactly ones, and return to the starting city. The goal is to find the shortest possible route that visit all city exactly ones and return to the orgin.

* Using branch and bound method

(1) Start with one city and explore all possible routes to other cities.

(2) But instead of checking all path we,

- calculate a minimum possible cost (called bound) for each path

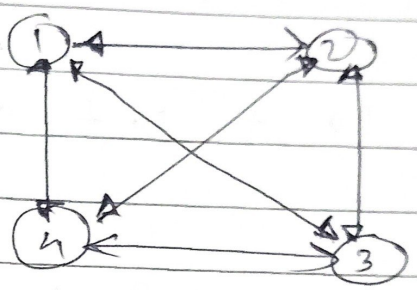
- Skip path that already cost more than the best solution we have so far.

(3) Always explore the cheapest - looking path using a priority queue.

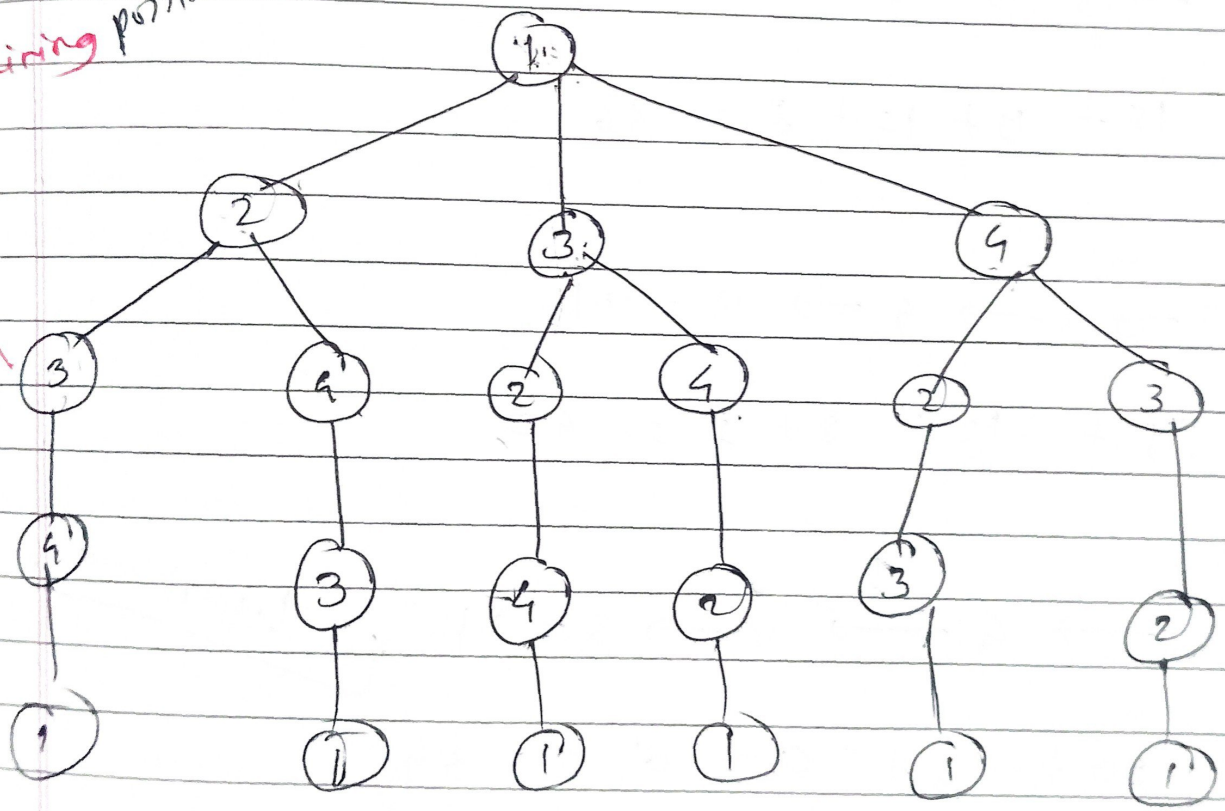
the origin.

eg:

	1	2	3	4
1	0	10	11	20
2	5	∞	9	10
3	6	13	0	12
4	8	8	9	0



remaining possibilities



All possible ways

① $1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 1$

Cost $10 + 9 + 12 + 8 = 39$

② $1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 3 \longrightarrow 1$

Cost $10 + 10 + 9 + 5 = 35$

③ $1 \longrightarrow 3 \longrightarrow 2 \longrightarrow 4 \longrightarrow 1$

Cost $15 + 13 + 10 + 8 = 46$

④ $1 \longrightarrow 3 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1$

Cost $15 + 12 + 8 + 5 = 40$

⑤ $1 \longrightarrow 4 \longrightarrow 2 \longrightarrow 3 \longrightarrow 1$ Lowest cost

Cost $20 + 8 + 0 + 6 = 34$

⑥ $1 \longrightarrow 4 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1$

Cost $20 + 9 + 13 \rightarrow 5 = 47$

AOA UT-2

Q) Naive string Matching algorithm ?

Ans:

The naive string matching algorithm is the simplest method for searching a pattern string within a text string.

It checks for the pattern at every posible position in the text, without using any preprocessing or optimization.

This algorithm follows the brute for approach

Step1: Let

T = text of length n

P = pattern of the length m

① start at index i = 0 to the next

② for each position i from 0 to n-m;

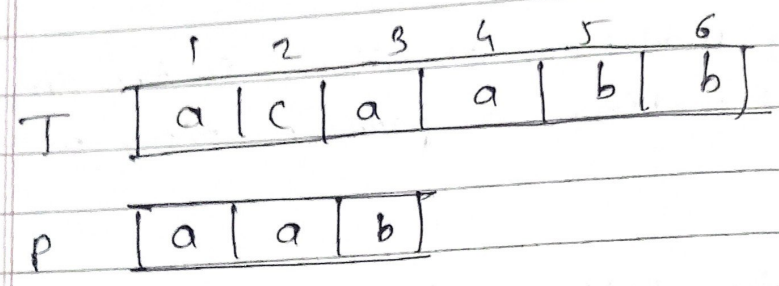
③ - compare p[0---m-1] with T[i---i+m-1]

- if all character match, 'H's match

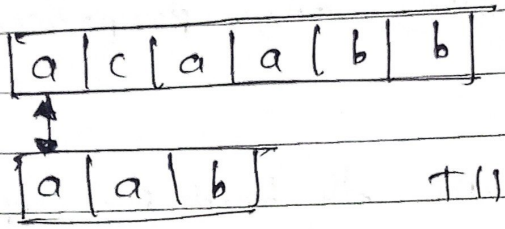
- if not move to the next position (i+1)

example: T = a c a a b b

p = a a b

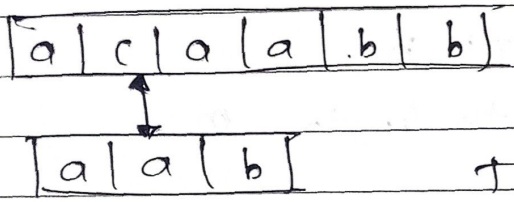


Step 1:



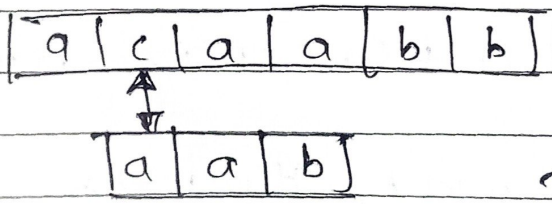
T[1] = P[1] then mov next pointer ptt

Step 2:



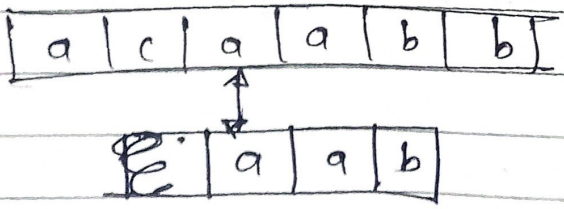
T[2] ≠ P[2] then mov text pointer tt+ and decrease p--

Step 3:



T[2] ≠ P[1] so tt+

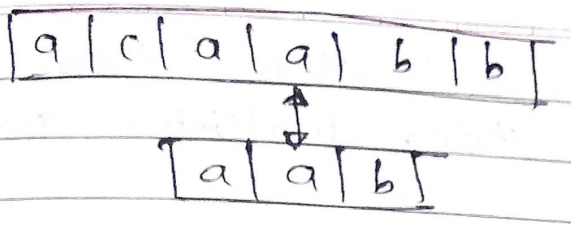
Step 4:



T[3] = P[1] so mov

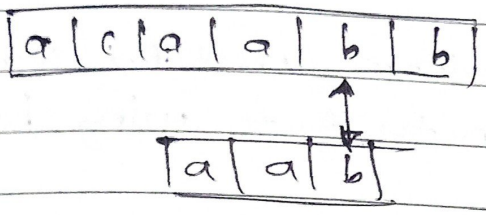
p++;

Steps:



T[4] = P[2] so p++;

Step 6:



T[5] = P[3]

match found at index 3 and 5

checks Remaining possibility in answersheet ....

Time complexity:

① Best case: O(n) → when pattern doesn't match often

② Worst case: O(n * m) → when pattern and text share many similar characters

PAGE No.
DATE / /

Q) What is Robin-Karp algorithm ? How is it better than Naive string matching algorithm. Write its TC & SC

→

- Robin-Karp is an algorithm used for searching a matching pattern in the text using hash function.

- How is it better than Naive?

Instead of comparing characters one by one at each step/position (as in the naive algorithm)

Rabin Karp compares hash values of strings. This makes it much faster in practice, especially when searching for multiple patterns.

eg:

1	3	1	1	2	1	1
a	c	a	a	b	a	a

pattern = aa

a = 1
b = 2
c = 3

aca = 5
caa = 5
aab = 4
aba = 4
baa = 4

from above string matching there are 3 string matching the hash value,

aab, aba & baa

but only aab match the pattern, so aab is the matching string.

★ time complexity (TC):

- Avg case: O(n+m)
- ~~Best~~
  worst case: O(n * m) → due to hash collision requiring character checks

★ space complexity: (SC):

- O(1) → single pattern
- O(k) → if multiple pattern

optional

(Q) Explain Knuth-Morris-Pratt algorithm with an example.

Ans:

- the KMP (Knuth-Morris-Pratt) is an efficient string matching algorithm that avoids unnecessary comparision by preprocessing the pattern.
- it uses prefix table (also called LPS array - longest prefix which is also suffix) to skip characters while matching improving performance over the naive algorithm.

★ Algorithm:

1. define the prefix function.
2. Slide the pattern over the text for comparision
3. If all the characters are match, we found the matching string.
4. If not,
   we set a prefix function to skip unnecessary comparision.

If the LPS values of previous character from mismatched character is 'a' then start comparision from index 0 of pattern with next character in the text.

However if the LPS values is more than 0 Start the comparission from index value equal to LPS value of the privously mismatched character.

example:

a b c d a b e a b F

ind:	1	2	3	4	5	6	7	8	9	10
	a	b	c	d	a	b	e	a	b	F
	0	0	0	0	1	2	0	5	6	0

patten a b e a

occur again so put the index

4	2	0	5

→ string matches at ind → 5 to 8

* TC → O(m) → LPS building
O(n) → patten matching

Overall TC → O(n+m)

PAGE No.
DATE / /

Explain Floyd - Warshall Algorithm:
(All pairs shortest path)

→

- The Floyd-Warshall algorithm is a dynamic programming algorithm used to find the shortest distances between all pairs of vertices in a weighted directed graph (can also work for undirected graph

- It works for graphs with positive or negative edges weights, but no negative weighted cycle.

~~example:~~

Formula: $D^k[i,j] = \min\{ D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j] \}$

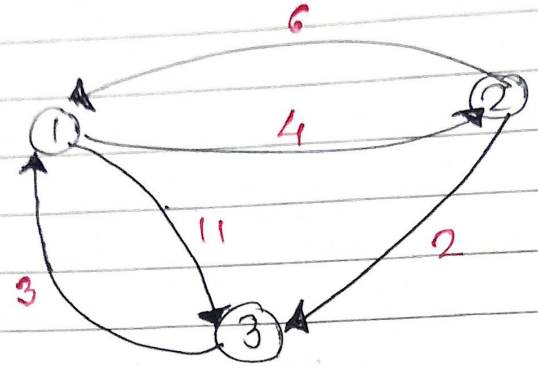
Where D^k = distance matrix after k^{th} iteration

Algo:

```
for k=0 to v-1
    for i=0 to v-1
        for j=0 to v-1

D[i,j]^k = min(D[i,j]^{k-1}, D[i,k]^{k-1} + D[k,j]^{k-1})

        end
    end
end
return 0;
```

k=1 first column and first row on it i) baki ka change

* if asked example then ,



	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

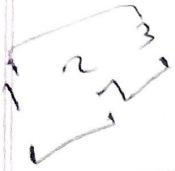
we used recurrence formula.

$$D^k[i,j] = \min\{ D^k[i,j],\ D^{k-1}[i,k] + D^{k-1}[k,j] \}$$

Iteration = 1 (k = 1)

	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

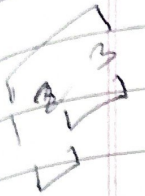
- 2 → 3
- 3 → 2



$D'[2,3] = \min\{ D^0[2,3) ; D^0[2,1) + [1,3] \}$

$= \min\{2, 6+11\}$

$= \min\{2, 17\}$



$D'[2,3) = 2$

$D'[3,2] = \min\{ D^0[3,2], D^0[3,1] + D^0[1,2] \}$

$\min\{\infty, 3+4\}$

$\min(\infty, 7)$

$\min = 7$ //

Iteration 2 $(k=2)$

	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$1 \to 3$, $3 \to 1$

$D^2[1,3] = \min\{ D^2(1,3], D'[1,2] + D'[2,3) \}$

$= \min\{11, 4+2\}$

$\min\{11, 6\}$

$= 6$ //

Now $D^2[3,1] = \min[D^1(3,1), D^1(3,2) + D^1[2,3]]$

$= \min(3, 7+6)$

$\min(3, 13)$

$\min = 3$

	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

Iteration 3:

$1 \rightarrow 2$

$2 \rightarrow 1$

$D^3[1,2] = \min\{D^3(1,1), D^2(1,3) + [3,2]\}$

$= \min\{4, 6+7\}$

$= \min = \{4, 13\}$

$= 4$//

Now $D^3[2,1] = \min[D^3[2,1], D^2[2,3] + [3,1]]$

$= \min\{6, 2+3\}$

$\min\{6, 5\}$

$\min = 5$//

PAGE No.
DATE / /

(5) explain 15 puzzle problem (may be optional)

Ans:

- The 15 puzzle problem is a classic sliding puzzle consisting of:

(1) A 4×4 grid (total 16 positions)

(2) 15 numbered tiles (from 1 to 15)

(3) 1 empty space (used to slide tiles)

The goal is to rearrange the tiles in order from 1 to 15, with the empty space in the bottom-right corner

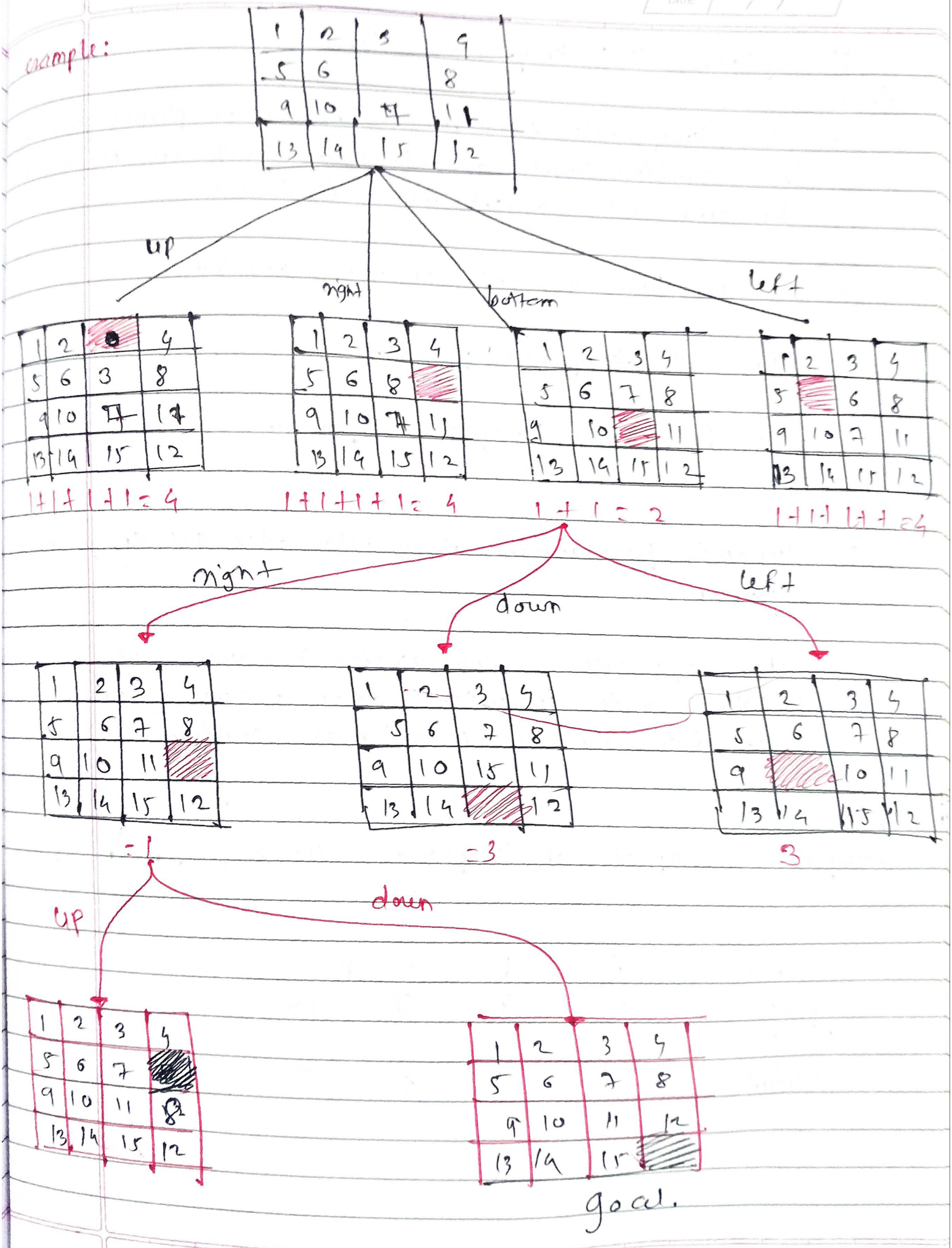
(6) How it's works?

(1) You can slide adjacent tiles (up, down, left, right) into the empty space.

(2) only one tile can be moved at a time.

(3) The challenge is to reach the goal configuration using the least number of moves.

example:


up
right
bottom
left
1+1+1=4
1+1+1+1=4
1+1=2
1+1+1+1=4
right
down
left
=1
=3
3
up
down
goal.

PAGE No.
DATE / /

Q) What is Backtracking and Branch & Bound? give 2 explains example of each. *

Ans:

* Backtracking: Backtracking is a method of solving problems by trying out all possible options and discarding those that do not satisfy the problem's conditions.

It builds solutions step-by-step and goes back (backtracks) when it hits a dead end.

Branch & Bound: Branch and bounds is an algorithm used for solving optimization problems

It divides the problems into subproblems (branching) and uses limits (bounds) to avoid exploring parts of the solution space that cannot give better results.

Branching: divide problem into subproblems

Bounding: Calculate upper / lower bounds to eliminate bad branches.

example of ~~branch & bound~~: branch & bound.

① 0/1 knapsack problem (optimal)

② Travelling salesman problem (TSP)

example of ~~Branch & Bound~~: Backtracking

① N-queens problem ④ graph coloring

② sudoku solver ⑤ 15-puzzle

③ sum of subset

Q. What is DP? two elements of DP? application of DP? ★

Ans:

- dynamic programming is a method for solving complex problem by breaking them down into smaller overlapping subproblem and solving each subproblem only once, storing the results for future use.

- It also known as tabulation or memoization

- Two elements of DP

① optimal structure: the soln to a problem can be built from the solution of its subproblem

② Overlapping subproblems: the problem can be broken into subproblem which are reused multiple times.

PAGE No.
DATE / /

⊙ Application of DP

① 0/1 knapsack problem
② Longest common subsequence (LCS)
③ coin change problem
④ shortest path Algorithm (Floyd-warshall, Bellman

⊙ Elements of DP

There are 3 elements that characterize a dp algorithm

① Substructure:
decompose the given problem into smaller problems.

② table structure:
after solving the sub problems, store the results to the subproblems in a table.

③ Bottom-up computation:
Using table, combine the soln for smaller sub problems to solve larger problem & eventually arrives at a solution to complete problem.

① Advantage of DP	① disadvantage of DP
- solves problems efficiently that have overlapping subproblems	- High space complexity (use large memory tables)
- Reduce TC by avoiding redundant calculation	- May be hard to design the recurrence relation
- Useful in optimal problems	- Not suitable if subproblem do not overlap.

② explain Sum of Subset using Backtracking.

Ans:

definition: The sum of subset problem is a decision problem where, given a set of positive integers and target sum S., we are asked to find all subset of the set whose elements add up exactly to S.

You must use backtracking to explore all possible subset, but stop exploring if the current sum exceeding the target.

★ Backtracking steps:

① Start from index = 0

② At each index, we choose or don't choose the current elements.

(3) if sum > target → backtrack

(4) if sum == target → print the current subset

example:

problem: Set = {2, 3, 5}
target sum = 5

Step ① start with element 2

include 2

currentSum = 2 , subset = {2}

Step ② Go to element 3

include 3

(2+3)

current sum = 5 , subset = {2, 3}

This valid soln, print it and remove it through back track.

Back tracks → remove 3

subset = {2} currentsum = 2

zemax → 2

currentSum = 0 subset { }

Step (2) Go to next elemt 5

include 5

currentSum = 5 subset = {5}

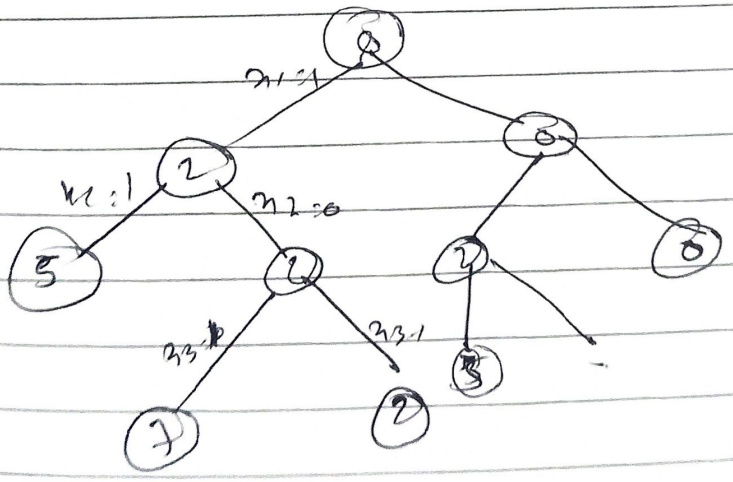
valid solution!

final answer:

{2,3}

{5}

(2, 3, 5)



(9) Sum of subset problem:

x_1 x_2 x_3 x_4

1. {10, 20, 30, 40}

sum = 50