Selection Sort

```
def greedy_selection_sort (arr):
    n = len (arr)

    for i in range:
        min_index = i
        for j in range (i+1, n):
            if arr[j] < arr[min_index]:
                min_index = j


def selectionSort (arr):
    for i in range (len (arr)):
        min = float ('inf')
        for j in range (i+1, len (arr)):
            if (arr[i] > arr[j]):
                arr[i], arr[j] = [arr[j], arr[i]
    return arr

arr = input ('Enter the list of numbers: ').split ()
arr = [int (x) for x in arr]

def selectionsort (arr)
    for i in range (len (arr):
        minindex = i
        for j in range (i+1, len(arr)):
            if (arr[j] < arr[minindex]):
                minindex = j
        arr[i], arr[j] = arr[j], arr[i]
    return(arr)
```

design tools can generate forms that could not Exist without computation.

(4) Moreover, AI makes fast effective and alternative method for visualization & prototype production possible.

(5) Building information modelling (BIM) Software aid architecture to handle design and construction stages as a holistic process

8/10

Selection Sort

```
def greedy_selection_sort (arr):
    n = len (arr)

    for i in range:
        min_index = i
        for j in range (i+1, n):
            if arr[j] < arr[min_index]:
                min_index = j


def selectionSort (arr):
    for i in range (len(arr)):
        min = float ('inf')
        for j in range (i+1, len(arr)):
            if(arr[i] > arr[j]):
                arr[i], arr[j] = [arr[j], arr[i]
    return arr

arr = input ('Enter the list of numbers: ').split()
arr = [int (x) for x in arr]

def selectionsort (arr)
    for i in range (len(arr):
        minindex = i
        for j in range (i+1, len(arr)):
            if (arr[j] < arr[minindex]):
                minindex = j
        arr[i], arr[j] = arr[j], arr[i]
    return(arr)
```

① DFS

Depth first Search : It is a recursive & graph traversal algorithm that uses the backtracking principle searching all the vertices of a graph or tree. It makes use of stack. First it asks for a root node then it push all the adjacent node of that particular node into the stack and after once all the vertices are covered then it pops them from the stack.

LIFO

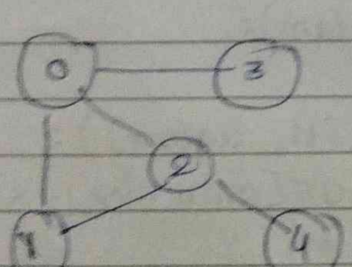
Recursive Algorithm : It makes use of Recursion. It is a programme technique where a function calls itself within its own defn. It simplifies a problem by breaking it down into sub-problems of the same type. The output of one recursion becomes the input for another recursion.

Types

① Direct Recursion - continually calls itself in the same block.

② Indirect Recursion - When a function invokes another function.

~~③ Tailed Recursion :-~~

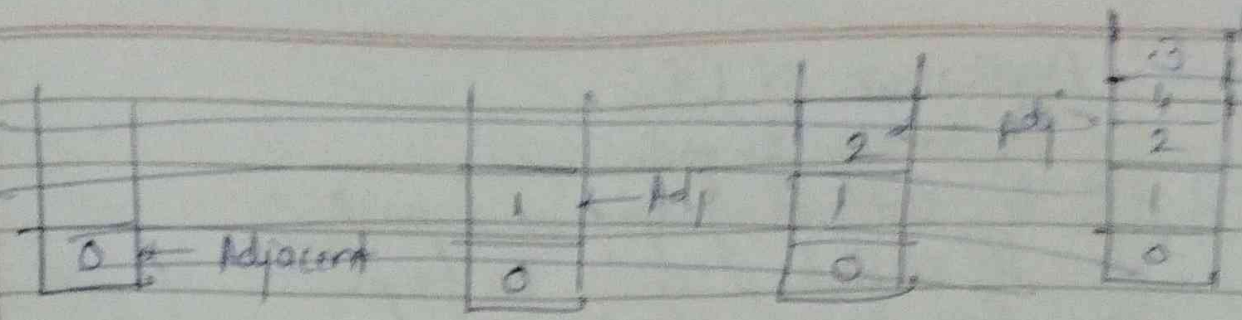


Input to be given

① 0 - 1
0 - 2
0 - 3
1 - 2
2 - 4

Enter the starting index for traversal : 0
DFS traversal 0, 1, 2, 4, 3

vewsal



Visited

A standard DFS puts each vertex of the graph or tree into 2 categories

Visited

Not Visited.

1. Start by putting any 1 of the graph vertices on the top of the stack
2. Take the top item of the stack and add it too visited list
3. Create a list of that vertex adjacent nodes
4. Add the ones which arent in the visited list to the top of the stack.

Visited | 0 |

V | 0 | 1

Adjacen Stack | 1 | 2 | 3 |

S | 2 | 3

V | 0 | 1 | 2

S | 4 | 3

V | 0 | 1 | 2 | 4

S | 3

V | 0 | 1 | 2 | 4 | 3 |

S |

Time Complexity = $O(V + E)$ → Vertices, Edges

Space Complexity = $O(V)$

286 → 296 → 306
20-122

- Application of DFS.
1. For finding the path
2. ~~To test if graph is bipartite.~~
2. To detect cycle in graph.
3. For puzzles like maze.
4. Topological sorting - mainly used for scheduling algorithms.

Disadv -- No guarantee to find a minimal path
- It goes to ∞ so we have invented DLS

(*) Asymptotic Notations.
These are the mathematical tools used to Analyze the performance of algorithms understanding how their efficiency changes as their input size grows

1. Big-Oh Notation - (O)
2. Omega Notation (Ω)
3. ~~Θ~~ Theta Notatⁿ (Θ)

1) (O) → represent the upper bound on an algorithm, gives worst case time complexity $f(n) = O(g(n))$
2) (Ω) → lower bound of an algorithm, best case time complexity $f(n) = \Omega(g(n))$
3) (Θ) - Encloses functⁿ from above & below represents both ↑ and ↓ bound on an algorithm, avg case time complexity
$f(n) = \Theta(g(n))$

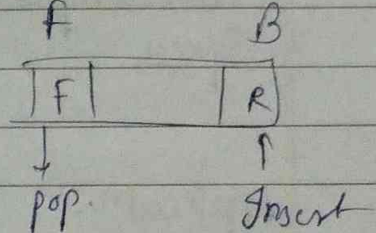
BFS

BFS is a graph traversal algorithm that explores all the vertices of a graph at a current depth before moving on to vertices at the next depth level. It starts at a specific vertex and visits all the neighbour before moving on to the next level of neighbours.

Graph traversal Algorithms refers to a process of visiting each graph in a vertex

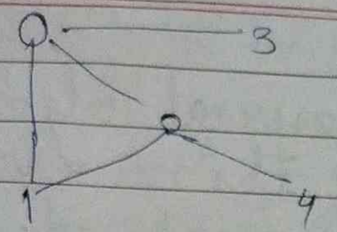
Ex → DFS BFS.

It makes uses of Queue.



Here it make use of two list Visited & Non visited.
Same steps as of DFS.

① Start by putting any one of the vertex at the back of a queue.
② Take the front item of queue and put it into visited list
③ Create a list of vertex's adjacent nodes. Add the ones which aren't in the visited list to the nodes.
④ Keep repeating steps 2 and 3 until the queue is Empty



| 0 | — V

| 1 | 2 | 3 . — Queue

front

V | 0 | 1

A

Q | 2 | 3

| 0 | 1 | 2

Adjacent 4

| 3 | 4 ←

R → push

| 0 | 1 | 2 | 3

| 4 |

V | 0 | 1 | 2 | 3 | 4

Q |

Time Complexity = $O(V+E)$
Space = $O(V)$

Applicatn.

1. Routing Algorithms
2. path finding in maps
3. Search Engine Crawlers
   - They are search Engines which helps to find, read and store pages to show.
4. Social Networking Websites

~~Adv~~

5. Minimum spanning tree for weighted graphs
6. Crawlers in Search Engine
7. GPS Navigatn System - to find locatn
8. Topological Sorting

~~DFS. - It comes under uninformed search Strategy. It is Expanded depth wise, i.e. deepest node in the current branch of the search tree. Then it backtracks again to previous node of a search tree.~~

Adv - BFS

1. BFS never gets trapped.
2. If there is a soln BFS will definetely behind it.
3. ~~Low storage requirement - linear with depth~~

Disadv BFS

1. Higher memory requirement as compared to DFS.
2. ~~May not always find shortest path.~~
3. ~~May get stuck in a local minimum in a weighted graph~~

*

Adv - DFS

Memory - Efficient.

Time Efficient - faster than BFS.

Disadv

1. Completeness
2. Non-optimal soln
3. Local minimum

No Backtracking Required	Required
BFS (Breadth)	DFS
(1) Queue	Stack
(2) FIFO	LIFO
(3) We walk through all the nodes on same level before moving on to next level	(3) Traverse begins at the root node and proceeds through the nodes as far as possible.
(4) suitable for searching vertices closer to the given vertices	(4) Suitable for sol^n which are away from source.
(5) Various Applicat^n Bipartite Graph Shortest path	(5) Acyclic graph
(6) More memory	(6) Less memory.

A* star Algorithm.

- Informed search technique. Search is based on Evaluat^n funct^n f(n). Evaluat^n funct^n is based on both Heuristic funct^n g(n) and h(n)

n = nodes

$$f(n) = \underbrace{g(n)}_{\text{cheapest cost (root node to node)}} + h(n) \longrightarrow \text{cheapest cost to reach from node to goal node}$$

It is a greedy best first search algorithm which combines heuristic approaches to find the shortest path

Example of A* Algorithm is N-Queen

Cost funct^n - The cost function Evaluates the cost of a state. in the N-Queen problem, the cost of a state is the number of pairs of queen that threaten Each other

Informed Search techniques - These are the techniques which known start and End state also it has an option to choose the alternate path

Heuristic funcⁿ - it estimates the cost of reaching the goal from a given state. Here a common heuristic is to count the pairs of queen that threaten Each other in the current state

A* Search - It uses both the cost function and the heuristic funcⁿ to guide the search towards the goal.
A* select the state with the lowest combined cost and heuristic value and Explores its successor until a goal state is found.

A* is a graph traversal technique which is widely used in AI to find the shortest path between two nodes
Makes use of Best first Search

A* Search makes use of 2 queues
→ Open → priority queue → ascending order
Close

Completeness ⇒ Complete
optimality ⇒ optimal
Time Complexity ⇒ ~~O(bm)~~ $O(b^m)$
Space Complexity ⇒ ~~O(bm)~~ $O(b^m)$

b = branching factor
& m ⇒ maximum depth of search tree

Greedy - Greedy search is an algorithmic approach for solving optimization problem.

In greedy algo, at each step of the problem solving process, it tries to find the locally optimal solution ~~but~~ with the hope that it will leads to a globally optimal sol^n.

Ex:

Arr = [2 , 5 , 1 , 7 , 8]

step 1: 2 , 5 , 1 , 7 , 8

swap

Step 2 : 1 , 5 , 2 , 7 , 8

3 : 1 , 2 , 5 , 7 , 8

4 : 1 , 2 , 5 , 7 , 8

5 : 1 , 2 , 5 , 7 , 8

6 : 1 , 2 , 5 , 7 , 8

Selection sort takes the first Element and compares it with the unsorted part and if it finds the smallest no. then it swaps the number with it.

→ simple algorithm divides list in 2 parts, sorted and unsorted one

Time complexity - $O(n^2)$ → because it iterates over the list repeatedly making multiple comparison

Space complexity - $O(1)$

↳ ~~it makes use of list~~

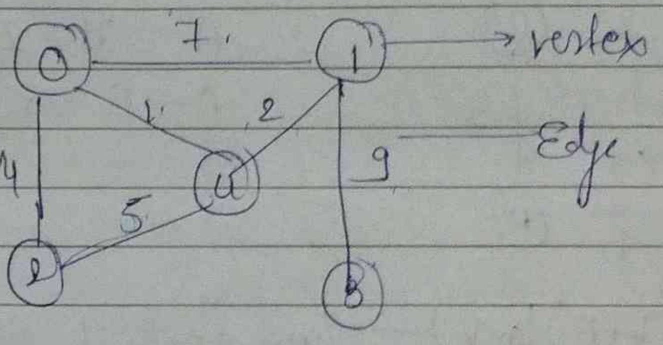
it sorts the list in place, hence requires a constant amount of space regardless of input size.

2) ~~Minimum~~

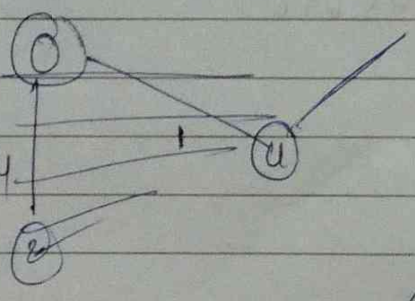
2) Minimum spanning Tree.

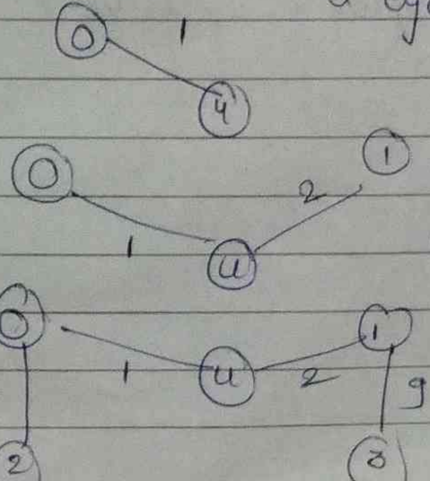
connected undirected graph, which makes use of a subgraph that is a tree and connects all vertices together with the minimum possible total edge weight.

It includes Kruskal & Prime



Kruskal.
- sorts all algorithms in their increasing order of weight without forming a cycle.





minimum weight. = 16

Time Complexity = $O(E \log E)$
E = number of Edges in graph

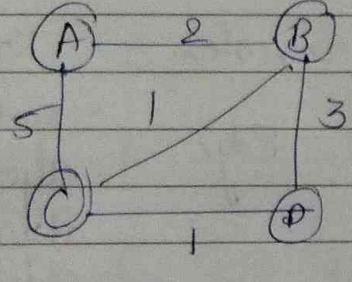
Space complexity $O(E)$

S'

single shortest path

It aims to find the shortest path from a single source vertex to all other vertices in a weighted graph

use priyush
Ex- dijkstra's Algorithm



A — A = 0
A — B = 2
A — C = 5

Dijkstra - $O((V+E)\log V)$ → Time complexity.
S Space - ~~O(VE)~~ $O(V+E)$

classmate
Date
Page

Job Scheduling

- It involves determining the order in which a set of tasks or jobs should be Executed on a set of resources ~~which~~ while optimizing certain objectives such as minimizing completion time, maximizing throughput or minimum resource utilizatn.

current time = 0

start time = max (current time, job id)

completion time = start time + processing time

current time = completion time.

	job id	processing time	start time	Completion tim
4	1	4		
2	2	3	5	8
1	3	2	3	5
5	4	6		
3	5	3		

job id = 3

1) current time = 0

start = max (0, 3) = 3

Completion = (3 + 2) = 5

current = 5

preference

1. processing time less
   If same then
2. job id less

job id = 2

1) Current = 5

~~max~~ start = max (5, 2) = 5

Completion = (5 + 3) = 8

current = 8.

job id = 5

current = 8

start = max (5, 8) = 8

Completn = 8 + 3
= 11

Current = 11

job id = 1
current = 11
start = max (11, 1) = 1
complet^n = (11 + 4) = 15
current = 15
~~job ic~~

job id = ~~6~~ 4
current = 15
start = max (15, 4) = 15
complet^n = (15 + 6) = 21
Completion Time.

Total Complet^n time = Completion Time - job id
= 60 - 15
= 45

It is Greedy Job scheduling algorithm specifically the Shortest Processing time (SPT) rule.

Here job are sorted in ascending order based on their processing time and if its same then on job id
It aims to schedule the jobs based on their processing time ~~in~~ ~~a~~ to minimize the total completion time

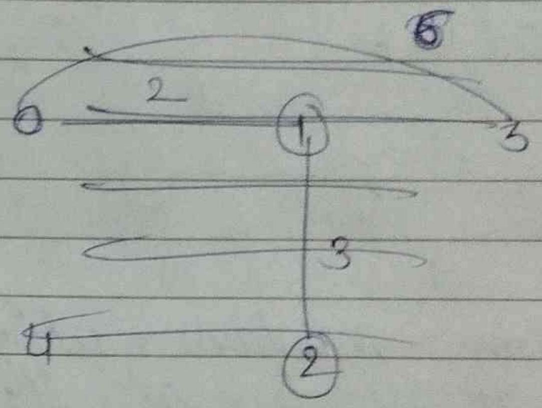
Time Complexity - O(nlogn) → dependent on sorting
Based on processing time & job id.

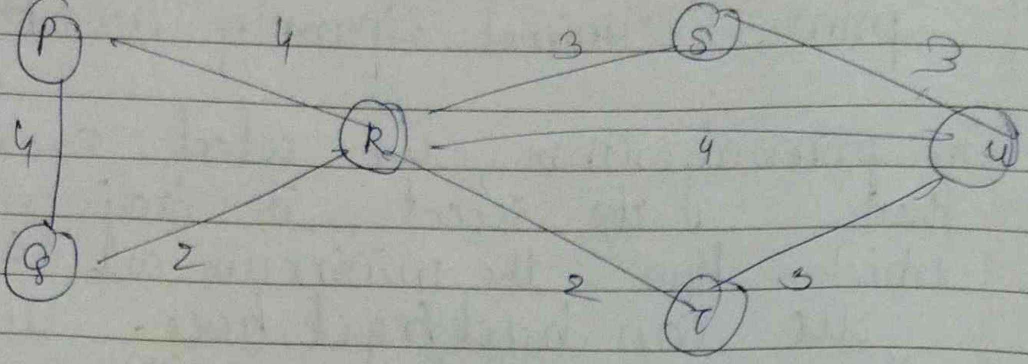
~~Pro~~ Space Complexity O(n)
→ because of 1 loop.
current time, total complet^n time, and loop counters require constant space.

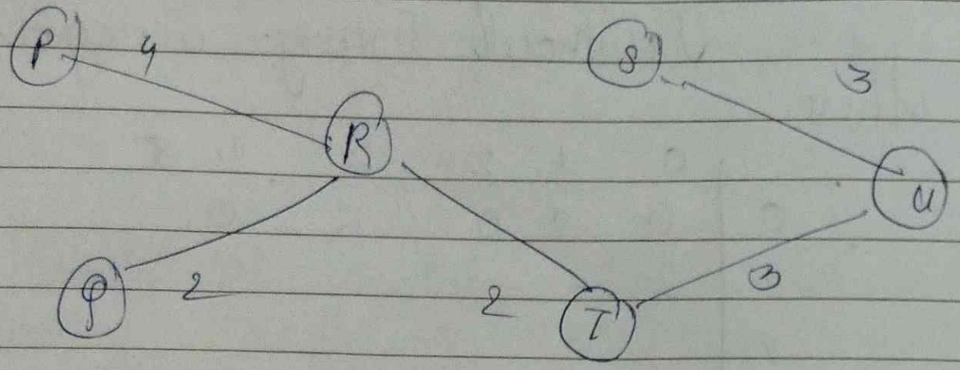
prim's Minimal Spanning Tree Algorithm

In prims algorithm we select a vertex then we select it adjacent vertex which has the minimum cost also we can backtrack here. and in this way we find minimum cost without forming a cyclic graph there

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

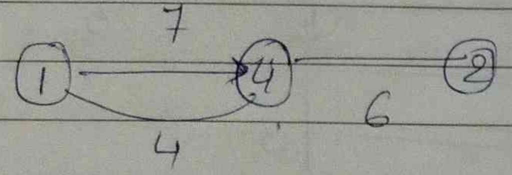




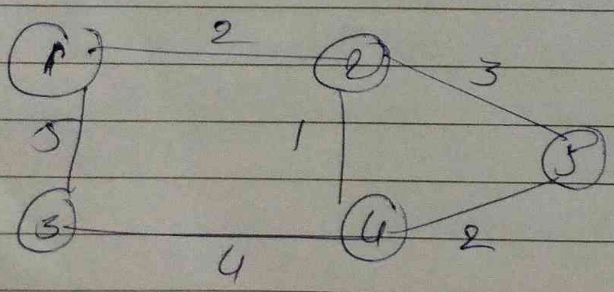


Time Complexity = $O((V+E) \log V)$

Space = $O(V+E)$



This algorithm maintains a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex and updates their distance if a shorter path is found



1 — 0
2 — 2
3 — 5
4 — 3
5 — 5

Time = $O((V+E)\log V)$

Space = $O(V+E)$

N Queen solut^n $\frac{N!}{(N-2)! \times 2}$ N = No. of Queen

$= \frac{8!}{(8-2)! \times 2} = \frac{\overset{4}{\cancel{8}} \times 7 \times \cancel{6!}}{\cancel{6!} \times \cancel{2}}$

$= 28$

N Queen problem,
It is placing of N chess queen on an N×N chessboard so that no two queen threaten each other.
We make use of backtracking algorithm here.

The idea is to place the queen one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queen. In the current column, if we find a row for which there is no clash, then we backtrack.

Basically we need to ensure 4 things
① No 2 queen share a column, row, diagonal.

Constraint satisfaction problem - These are the problems that must be solved

within that particular constraints
For Ex Sudoku, N Queen, Colouring Graph

(T) Time → $O(N^N)$ → Backtracking Approach, Recursive calls

Space → $O(N^2)$ → 2D Array

Chatbot :- A chatbot is an artificial Intelligence based software developed to interact with humans in their natural language. These chatbots can really converse through their auditory or textual methods and they can effortlessly mimic human language to communicate with human being

Categories of chatbots

① Rule based chatbot - Gives answers based on a list of predetermined rules which was primarily trained.

② Self - Learning Chatbot - They can learn on their own with the help of technologies such as AI and ML they can train their own behaviours & instances. Smarter than Rule based chatbots

(3) Retrieval Based Chatbot - Works on predefined input and sets responses. Pattern or Questions is inserted it utilizes a heuristic approach to deliver the relevant response.

(4) Generative Chatbot - In this the source code is converted from the one language to another language

Microsoft's Cortana
Apple's Siri
Amazon's Alexa

Benefits of Chatbots	Disadvantages
(1) 24x7 Availability	(1) Need Analyzing
(2) Instant Answer to Queries	(2) Limited Natural Language processing Ability
(3) Support multilanguage	(3) Data Security Concerns
(4) Simple and Easy to use UI	
(5) Reduce Errors	

Time Complexity $= O(1) \rightarrow$ recursive $\rightarrow$ constant

Space Complexity $= O(N) \Rightarrow$ because user input is fixed. $(N) =$ maximum length of user i/p.

Web search $\rightarrow O(1)$