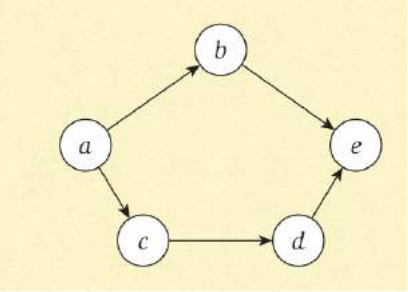
Solved Exercise 1

Consider the directed acyclic graph G in Figure 3.9. How many topological orderings does it have?



Solution Recall that a topological ordering of G is an ordering of the nodes as $v_1, v_2, \ldots, v_n$ so that all edges point "forward": for every edge (v_i, v_j), we have $i < j$.

So one way to answer this question would be to write down all $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ possible orderings and check whether each is a topological ordering. But this would take a while.

Instead, we think about this as follows. As we saw in the text (or reasoning directly from the definition), the first node in a topological ordering must be one that has no edge coming into it. Analogously, the last node must be one that has no edge leaving it. Thus, in every topological ordering of G, the node a must come first and the node e must come last.

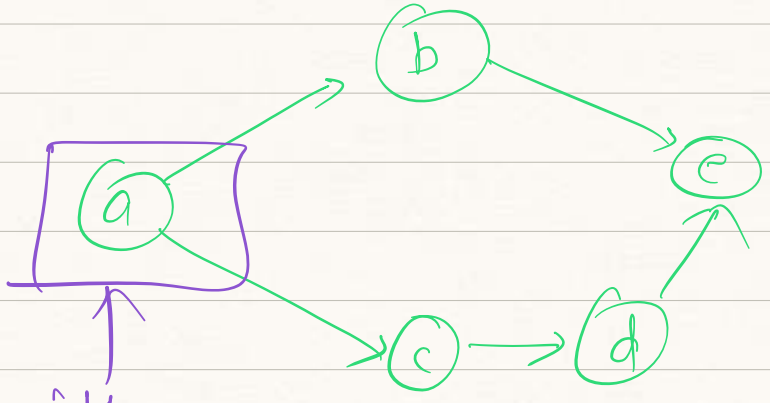
Now we have to figure how the nodes b, c, and d can be arranged in the middle of the ordering. The edge (c, d) enforces the requirement that c must come before d; but b can be placed anywhere relative to these two: before both, between c and d, or after both. This exhausts all the possibilities, and so we conclude that there are three possible topological orderings:

a, b, c, d, e

a, c, b, d, e

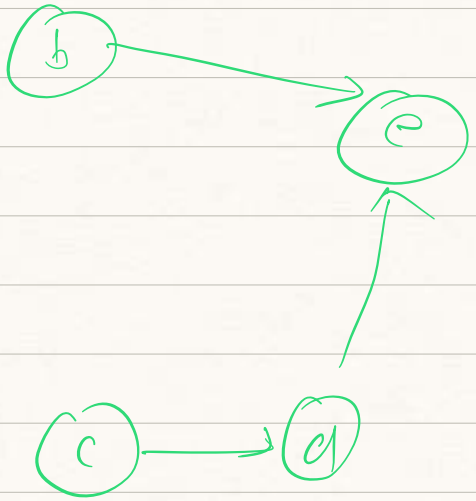
a, c, d, b, e

- In order to find topological order, first thing, find a node with no incoming edges



with
no incoming edge

- Now, remove all edges corresponding to Node - a



- Topological order possible

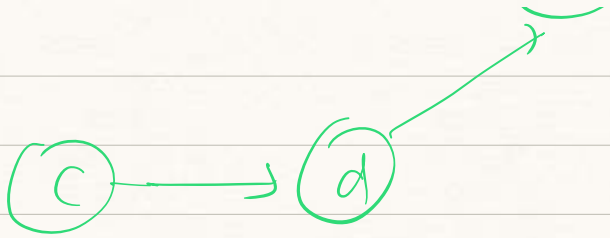
1. a - b - .....

2. a - c

1. a - b

- Remove node b & corresponding edges

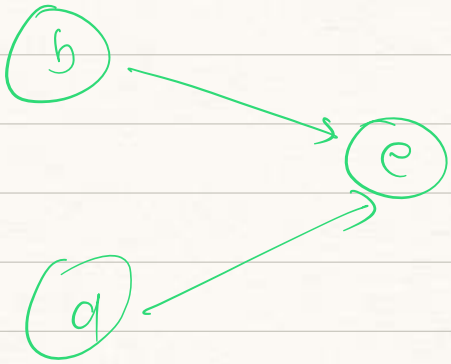
e



- Now, we can see that we don't have other possibilities to finish task - e from c
- Topological order
  a - b - c - d - e

2 (a) — (c)

- let's remove c & corresponding edges

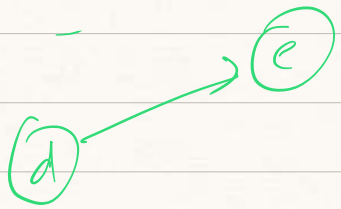


- Possible topological order

2-1 (a) — (c) — (b)

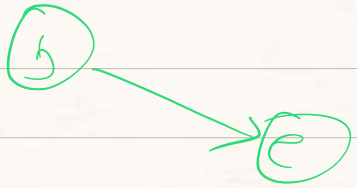
b-2 (a) — (c) — (d)

- 2-1 a - c - b



- a - c - b - d - e

- 2-2 a - c - d



- a - c - d - b - e

- Total possible topological order : 3
  - أ- a - b - c - d - e
  - ب a - c - b - d - e
  - ج a - c - d - b - e

Solved Exercise 2

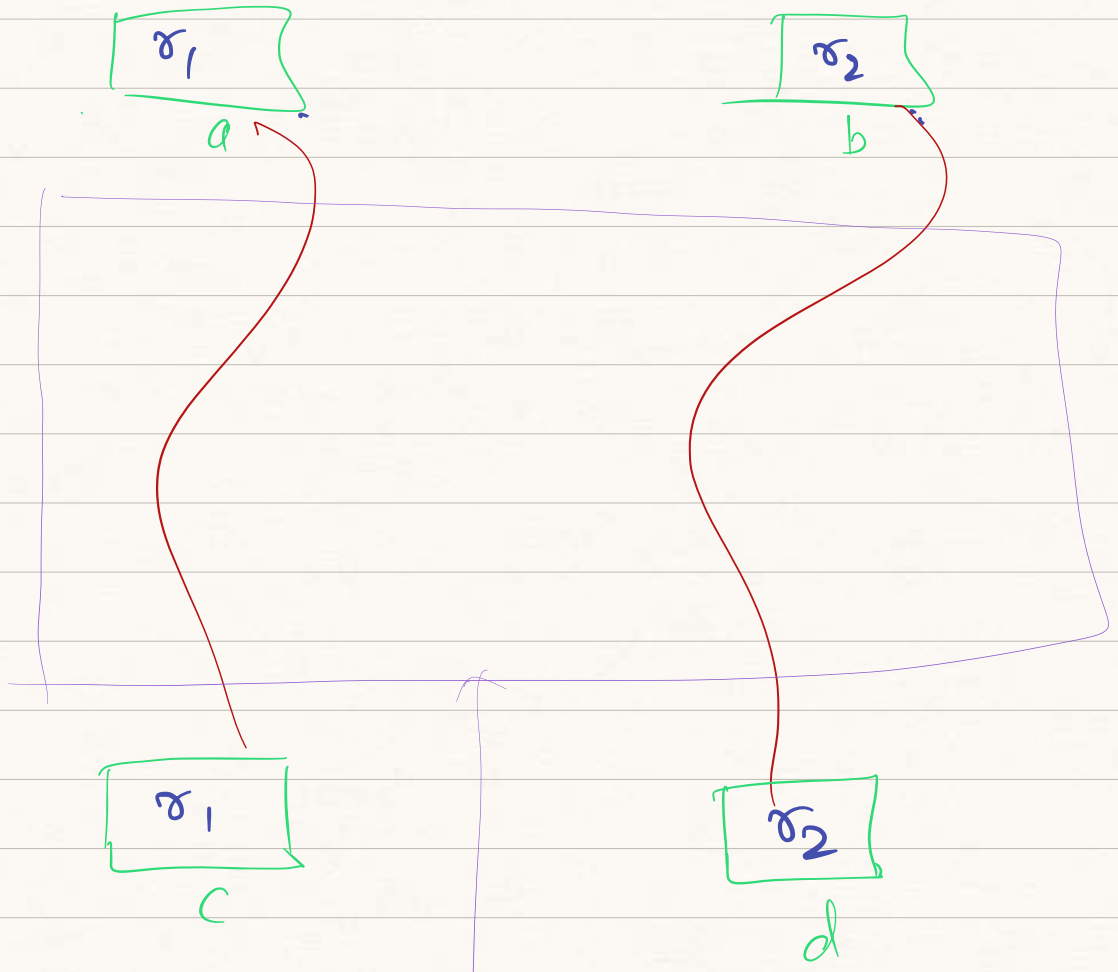
Some friends of yours are working on techniques for coordinating groups of mobile robots. Each robot has a radio transmitter that it uses to communicate with a base station, and your friends find that if the robots get too close to one another, then there are problems with interference among the transmitters. So a natural problem arises: how to plan the motion of the robots in such a way that each robot gets to its intended destination, but in the process the robots don't come close enough together to cause interference problems.

We can model this problem abstractly as follows. Suppose that we have an undirected graph $G = (V, E)$, representing the floor plan of a building, and there are two robots initially located at nodes a and b in the graph. The robot at node a wants to travel to node c along a path in G, and the robot at node b wants to travel to node d. This is accomplished by means of a *schedule*: at each time step, the schedule specifies that one of the robots moves across a single edge, from one node to a neighboring node; at the end of the schedule, the robot from node a should be sitting on c, and the robot from b should be sitting on d.

A schedule is *interference-free* if there is no point at which the two robots occupy nodes that are at a distance $\leq r$ from one another in the graph, for a given parameter r. We'll assume that the two starting nodes a and b are at a distance greater than r, and so are the two ending nodes c and d.

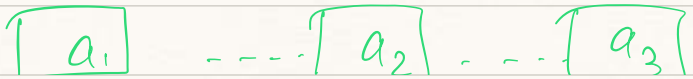
Give a polynomial-time algorithm that decides whether there exists an interference-free schedule by which each robot can get to its destination.

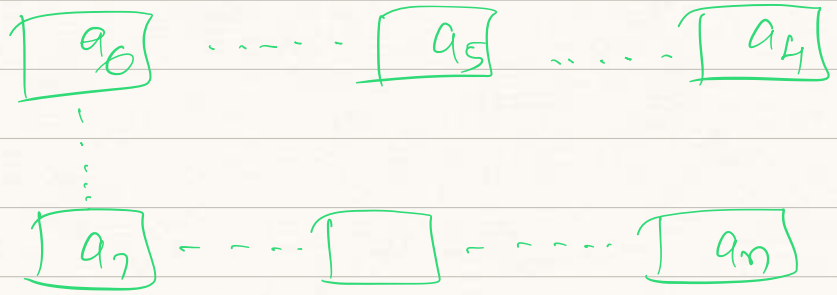
- This question is one of the most beautiful, I have seen recently
- Let's try to follow the solution given in the book
- Assume that we have two robots, initially located at



At any instance distance between path $\leq r$.
(In order to avoid interference.)

- Let's start solution.
  we have total n position





- 2 robots can initially take any 2 position (a_i, a_j) ← we will call it as configuration.

then

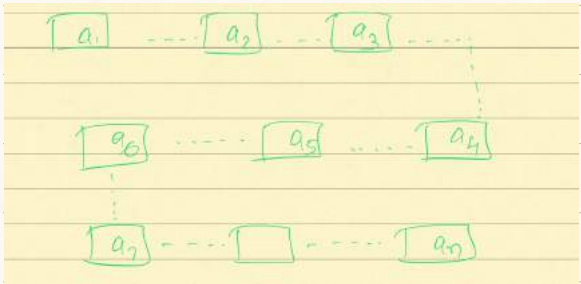
we want to find a path to destination (a_p, a_q)

destination configuration

- We want to find shortest path from source configuration to Destination configuration

But we have constraint to keep minimum distance b/w two robot $> \delta$

Let's call this



Graph - G

- Total possible combination

$^{n}C_2 \Rightarrow O(n^2)$

$\rightarrow \frac{n(n-1)}{2}$

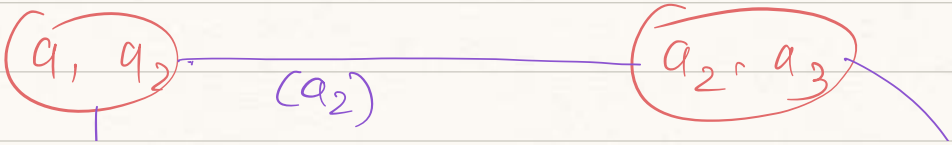
- Let's create a new graph of configuration, where each configuration is one node

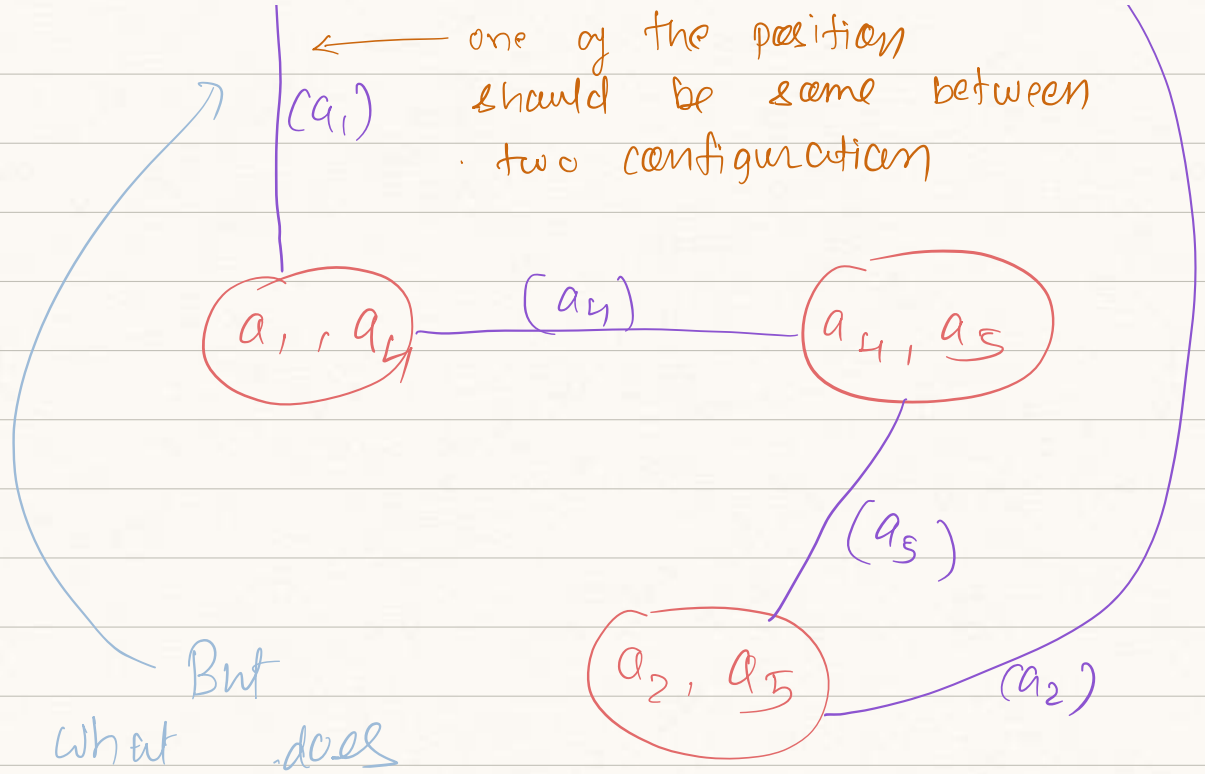
↳ total possible node

↳ $O(n^2)$

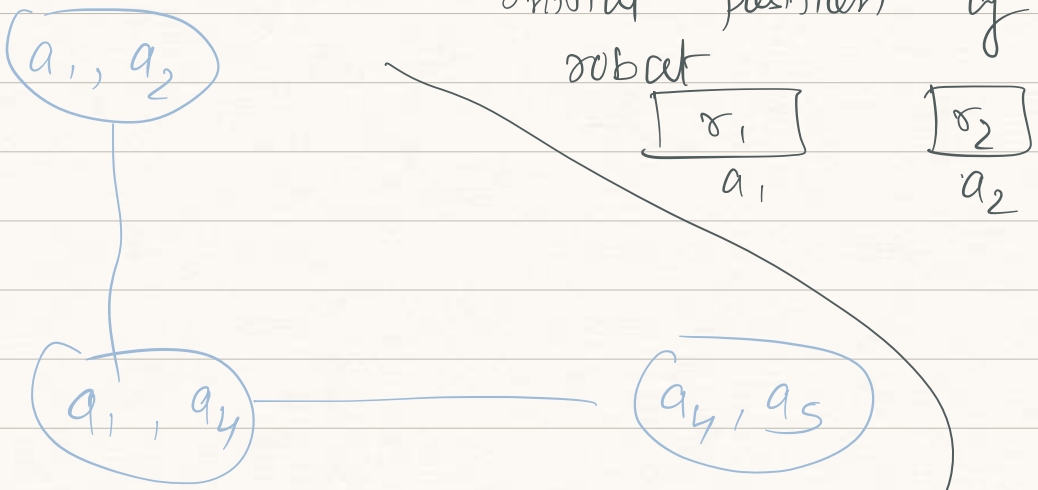
- What about edge 🤔 ,

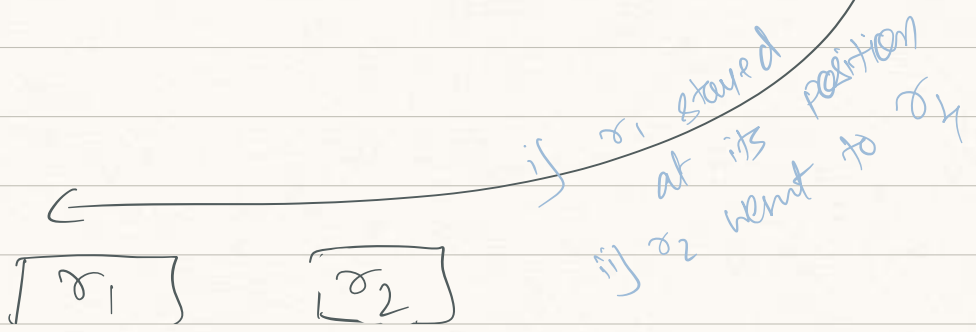
let's understand with example

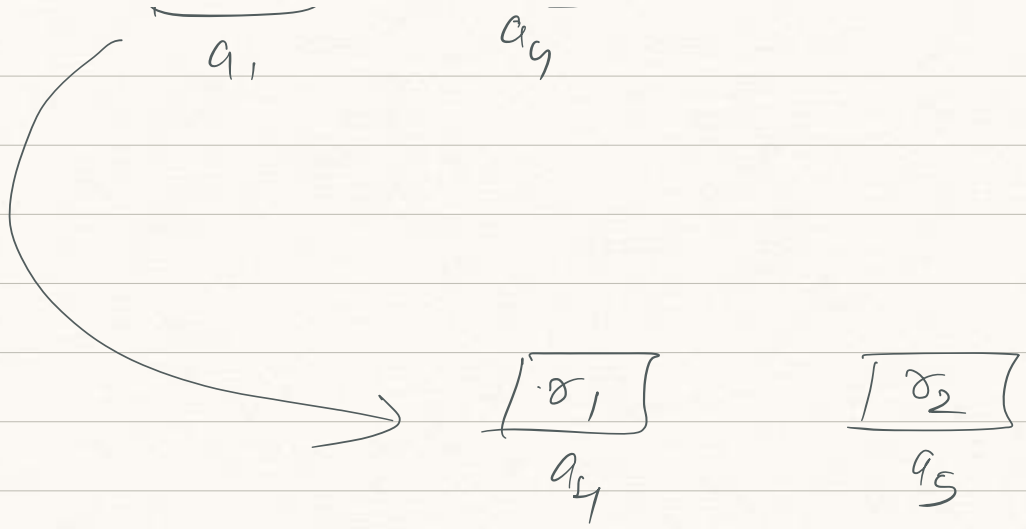




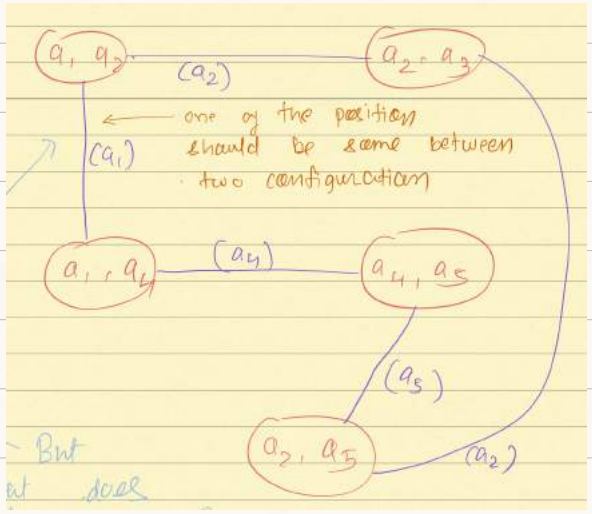
But what does this imply ?







Let's call this Graph



H

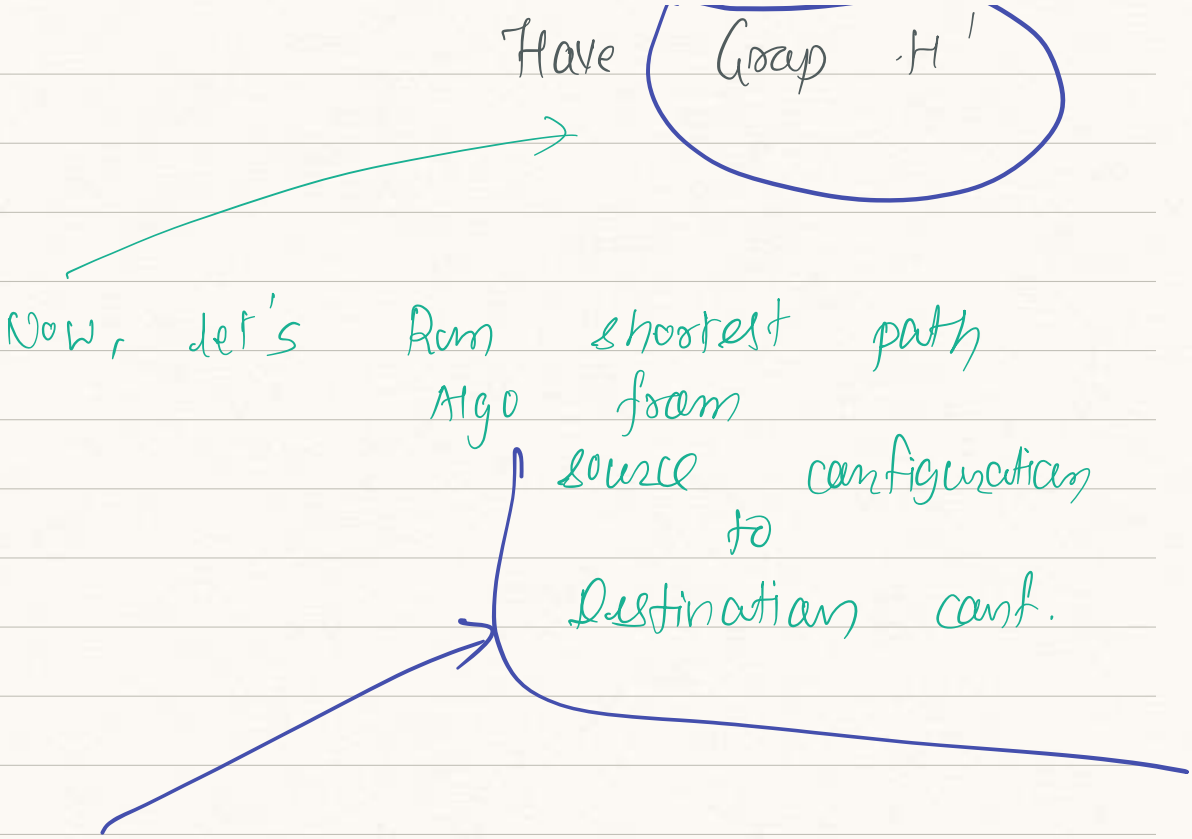
Now, Let's consider **CONSTRAINT**

≥

define H'

- Just remove all edges from H, that doesn't satisfy constraint

&



But

we are not finished yet.

- Complexity Analysis.

if n node in graph G

Graph H generation

#nodes = $O(n^2)$

- Possible edge from each node : $2n$

(Think why yourself)

#edges : $O(n^3) : O(E)$

ii) Constraint check $\Rightarrow O(E)$

iii) Shortest path :

$= O(V + E)$

where : $V = O(n^2)$

$E = O(n^3)$