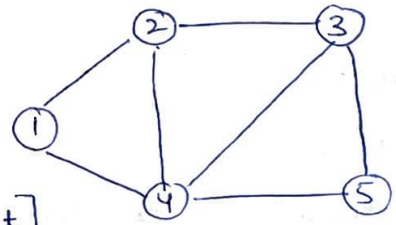
• Graphs

Adjacency Matrix :-

[It is a matrix A[n][n]
where, n is no. of vertices
& { a[i][j] = 1 if i & j are adjacent
= 0 otherwise]

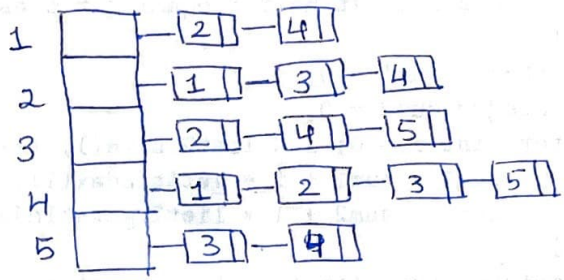


⇒

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

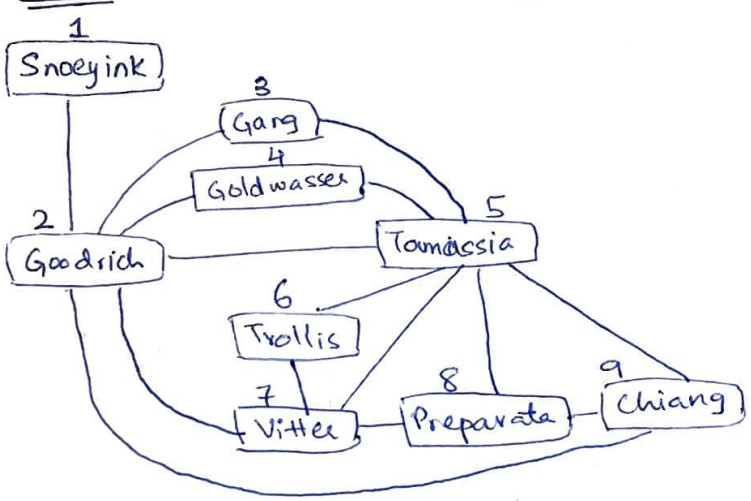
Adjacency List :-

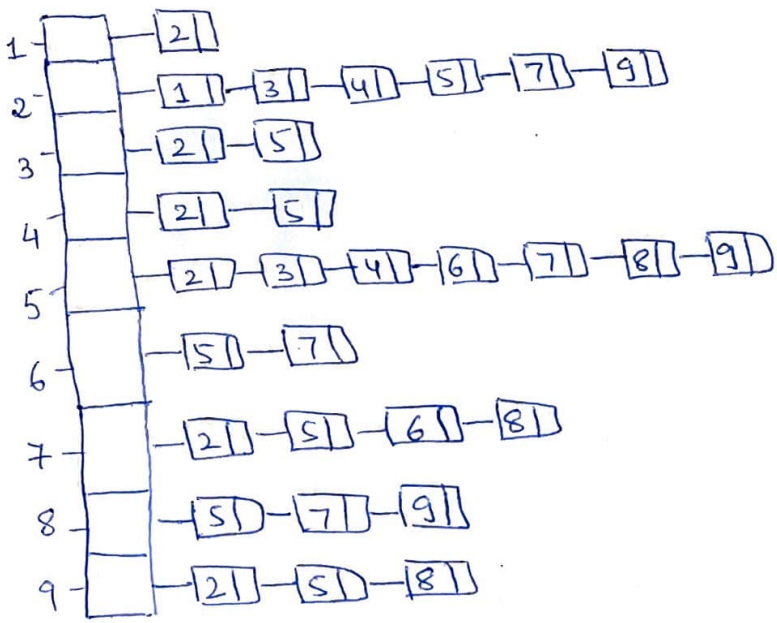
[For the same above graph] ⇒



Complexity → Adjacency List $O(V+E)$ Adjacency Matrix $O(V^2)$

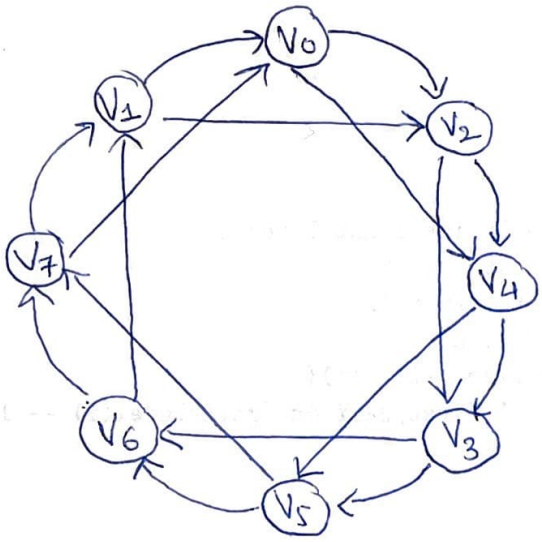
Q14.4 Draw Adjacency list for Fig 14.1 (undirected graph)





[You can also label all the edges and the draw the adjacency list as show in the book]

Q14.5 [8 vertices, 16 edges, in-degree = 2, out-degree = 2, Euler tour]



{This should be your final output}

[It satisfies all the conditions]

Q14.11

(a) Graph: 10,000 Vertices & 20,000 Edges

The graph is very sparse (only 2 edges per vertex on avg.)

Adjacency List ⇒ is a good option here as matrix would waste huge space (10,000 × 10,000 = 100M entries)

O(V+E)

(b) Graph: 10,000 Vertices & 20,000,000 Edges

This is a dense graph (almost full)

Both adjacent list & matrix are good but matrix here isn't wasteful and is manageable. Adjacency Matrix also gives quick constant-time edge lookup for later.

(c) How to getEdge(u,v) as fast as possible

✓ Adjacency Matrix
O(1) [search Time]

× Adjacency List
O(degree(u)) [slower]

Q14.38

[Here, to implement this you can use Hash Table, or Sorted ~~List~~ Structure or a Binary Search Tree (BST) that supports log n searches.]