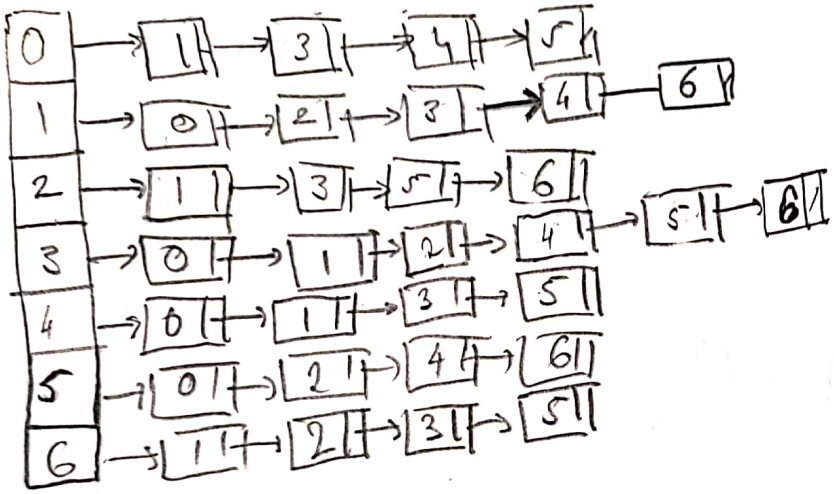
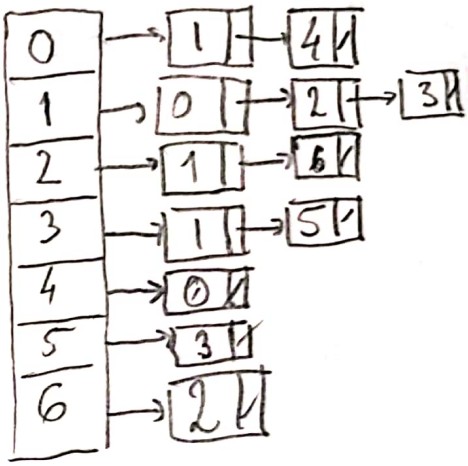
Q1

a) Adjacency list...

(i)



(ii)



b) Adjacency matrix

(i)

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

(ii)

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

Q 1

a) Each graph is an array of lists
Each vertex stores its edges inside of a linked list

(i)

arr entry	Edges
0	1 3 4 5
1	0 2 3 4 6
2	1 3 5 6
3	0 1 2 4 5 6
4	0 1 3 5
5	0 2 4 6
6	1 2 3 5

next step ⇒ just connect them

(ii)

arr. entry	Edges
0	1 4
1	0 2 3
2	1 6
3	1 5
4	0
5	3
6	2

next ⇒ just connect them using list

b) |V| x |V| grid (matrix) ← (i)
Since graph is undirected ← (ii)
⇒ symmetric graph

c)

① $|E| = 16$, $|V| = 7$

→ $\text{density} = \frac{2|E|}{|V|(|V|-1)} = \frac{2 \cdot 16}{7 \cdot (7-1)} = \frac{32}{7 \cdot 6} = 0.76$

⇒ $2|E|$ is less than $|V|^2 - |V|$ but not by a lot

→ this graph is **dense** ($|E|$ close to max $|E|$)

⇒ it is better to go with adjacency matrix. It saves a lot of memory and is faster in some cases.

- Constant edge searching operation
- Add new edge fast

② $|E| = 6$, $|V| = 6$

$\text{density} = \frac{2 \cdot 6}{6(6-1)} = \frac{12}{6 \cdot 5} = \frac{12}{30} = 0.28$

⇒ $2|E|$ is much less than $|V|^2 - |V|$

→ this graph is **sparse**

→ it is better to go with and use adjacency list. It is good for memory and algorithms are faster since not every 'possible' edge is checked.

- iterating over all nodes efficient
- Add/delete node is simple
- Adding new edge constant operation

(2) → root → p = [| | -1 | | | |] parent (p)

q: 2 Visited / identified used together simultaneously

v:

2 → 4 children now visited

6 5 3 1

0	1	2	3	4	5	6
|	2 |	-1 |	2 |	2 |	2 |	2

BFS

(1)

q: 6 5 3 1

v: 2

identified ⇒ p not updated

q: 6 5 3 1 (6) → 5 3 2 1

0	1	2	3	4	5	6
|	2 |	-1 |	2 |	2 |	2 |	2

v: 2 6

q: 5 3 1 → (5) → 6 4 3 2 0

identified

update

0	1	2	3	4	5	6
|	2 |	-1 |	2 |	5 |	2 |	2

q: 3 1 6 4

v: 2 5

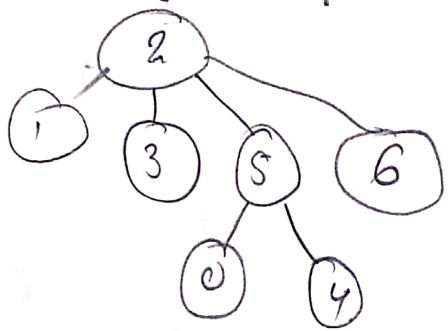
All vertices identified ⇒ would empty out queue without changing p

q:

v: 2 6 5 3 1 6 4 →

0	1	2	3	4	5	6
|	2 |	-1 |	2 |	5 |	2 |	2

↓ construct accordingly



BFS

par to sum

q: 2 6 1 6 1
v: 2

②

BFS

11

② → set adjacent vertices (1,6) parent to 2 P [| 2 | 1 | | | | 2 |]

q: 6 1 ⑥ → -//- use parent to 6 P [| 2 | -1 | | | | 2 |]
v: 2

q: 1 3 0 ① → -//- 3, 0's parent to 1 [| 1 | 2 | 1 | | | 1 | 2 |]
v: 2 6
add to q

q: 3 0 ③ → -//- 5's par is 3 [| 1 | 2 | -1 | 1 | | 3 | 2 |]
v: 2 6 1

q: 0 5 ⓪ → -//- 4's par to 0 [| 1 | 2 | -1 | 1 | 0 | 3 | 2 |]
v: 2 6 1 3

q: 5 4 → just pop since no children
v: 2 6 1 3

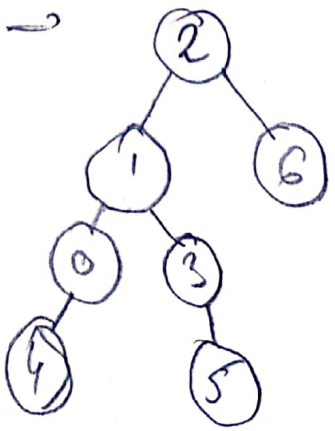
V: 2 6 1 3 5 4

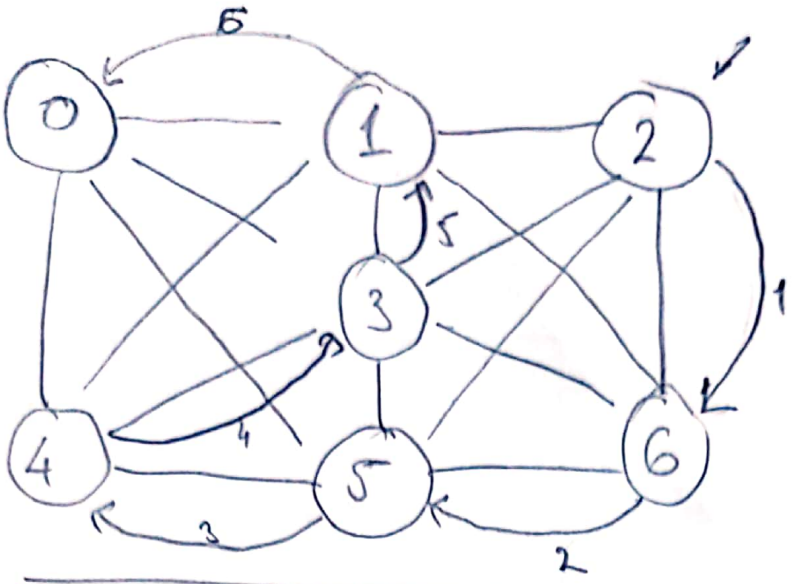
root

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

use it to construct a tree

→





DFS

0)

		-1				

1)

		-1				2

2)

		-1			6	2

3)

		-1		5	6	2

4)

		-1	4	5	6	2

5)

	3	-1	4	5	6	2

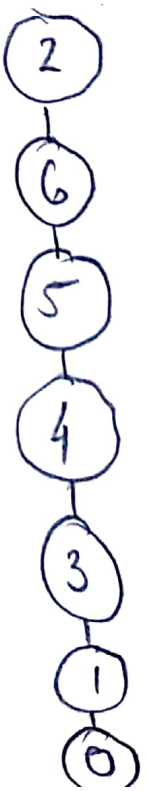
6)

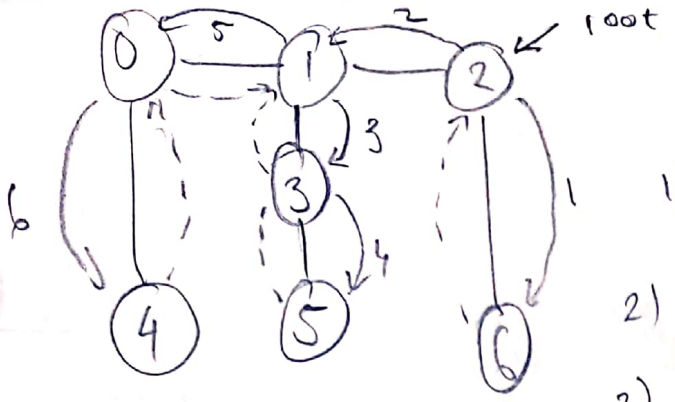
0	1	2	3	4	5	6	
1	3	-1	4	5	6	2	P

⇒

will start backtracking now but I won't include that part since it is not important for building a tree

Tree





		-1				

1)

		-1				2

2)

	2	-1				2

3)

	2	-1	1			2

4)

	2	-1	1		3	2

5)

1	2	-1	1		3	2

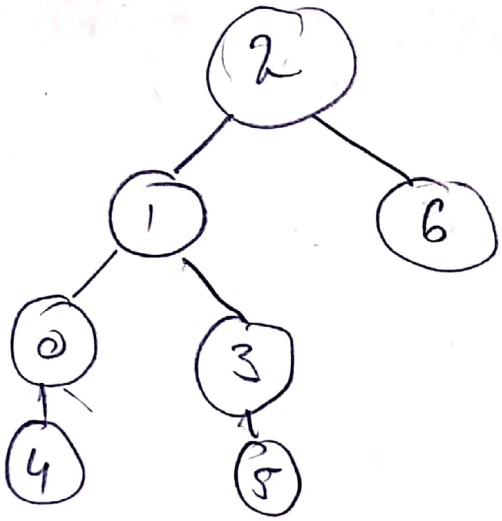
6)

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

DFS

⇒ Construct tree

⇒



All steps not included (back tracking) since they are not important for constructing a tree