Adjacency Matrix

Checking if there exists an edge between two nodes ? O(1).

Iterating through all neighbours ? O(|V|)

Space ? $O(|V|^2)$

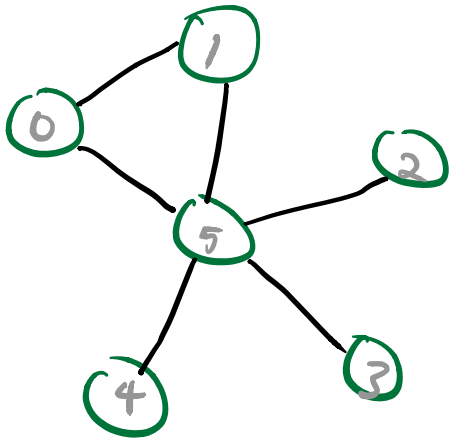
Adding an edge between vertices v and w ? connections[v][w]=1

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

undirected!

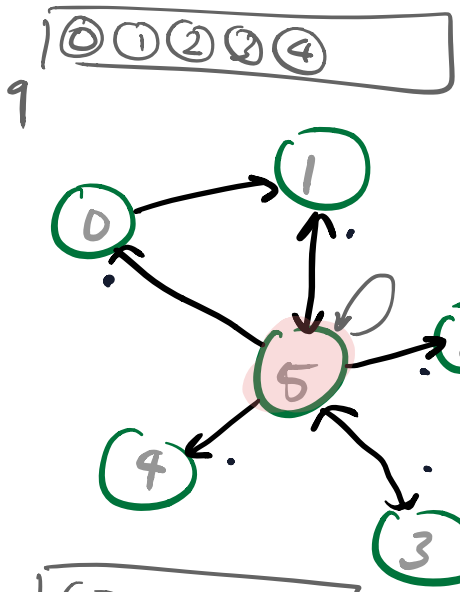
(u connected v → v connected to u)

"symmetric".



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

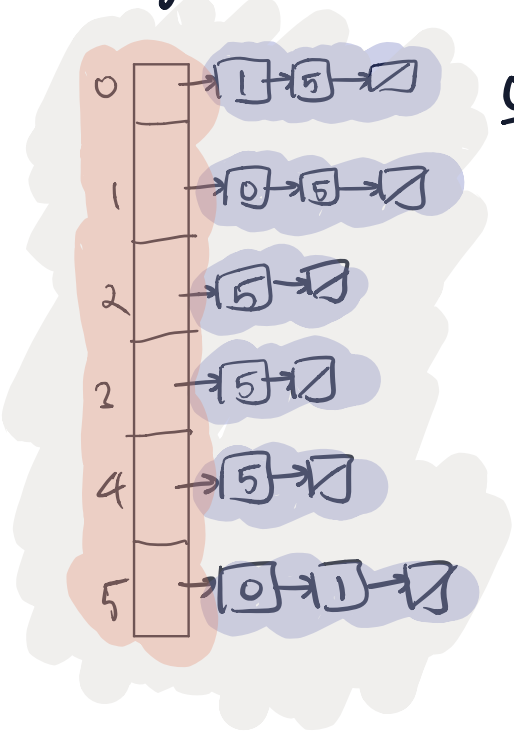
directed!



5

seen.

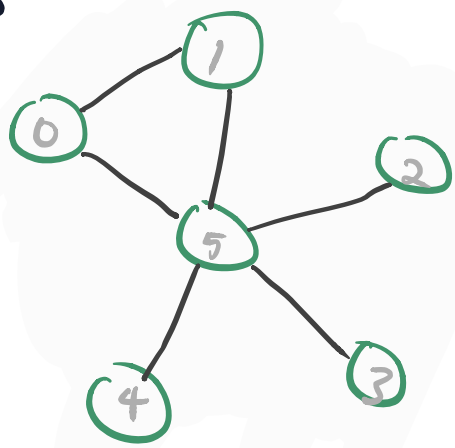
Adjacency List



Only store neighbours for a given vertex.

Space?

$O(|V|+|E|)$



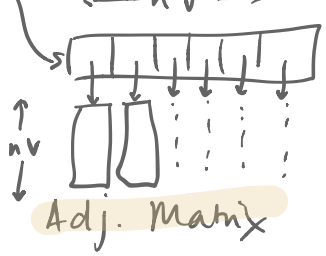
int nV
Adjlist *edges

struct adjlist :
Vertex node
Adjlist next

C Representation

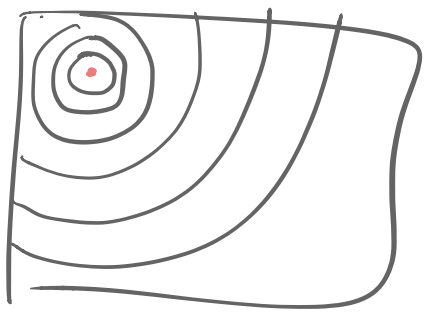
int nV
int ** edges
edges

not statically enforced!

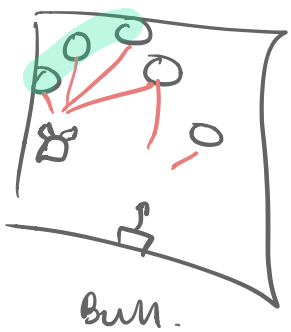


Adj. Matrix

Adj. List



ripple (BFS) vs. Bull. (DFS)



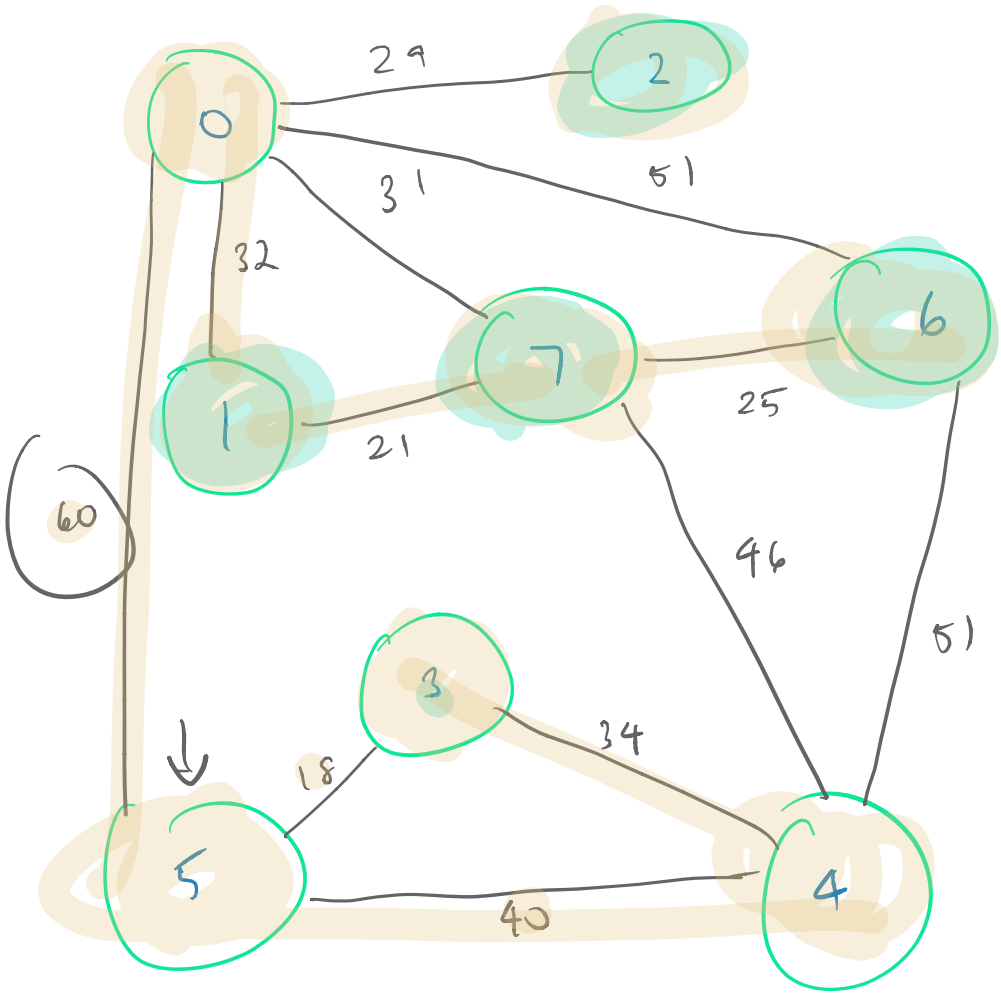
Questions.

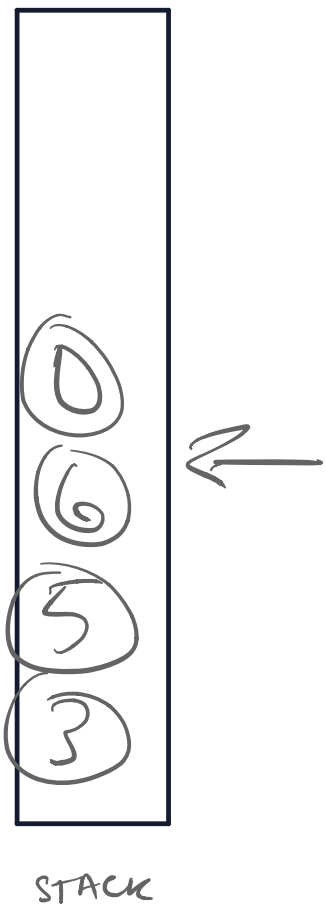
(a) Time complexity to check if two vertices are neighbours? Adding an edge?

(b) When is it appropriate to use an adj. matrix? similarly for adj list.

Extension: There is another representation of graphs that is useful in Functional Programming. Introduced in Erwig (2001) it's called an inductive graph. Inductive because it's defined in terms of itself.

Try and design an inductive graph.





DFS

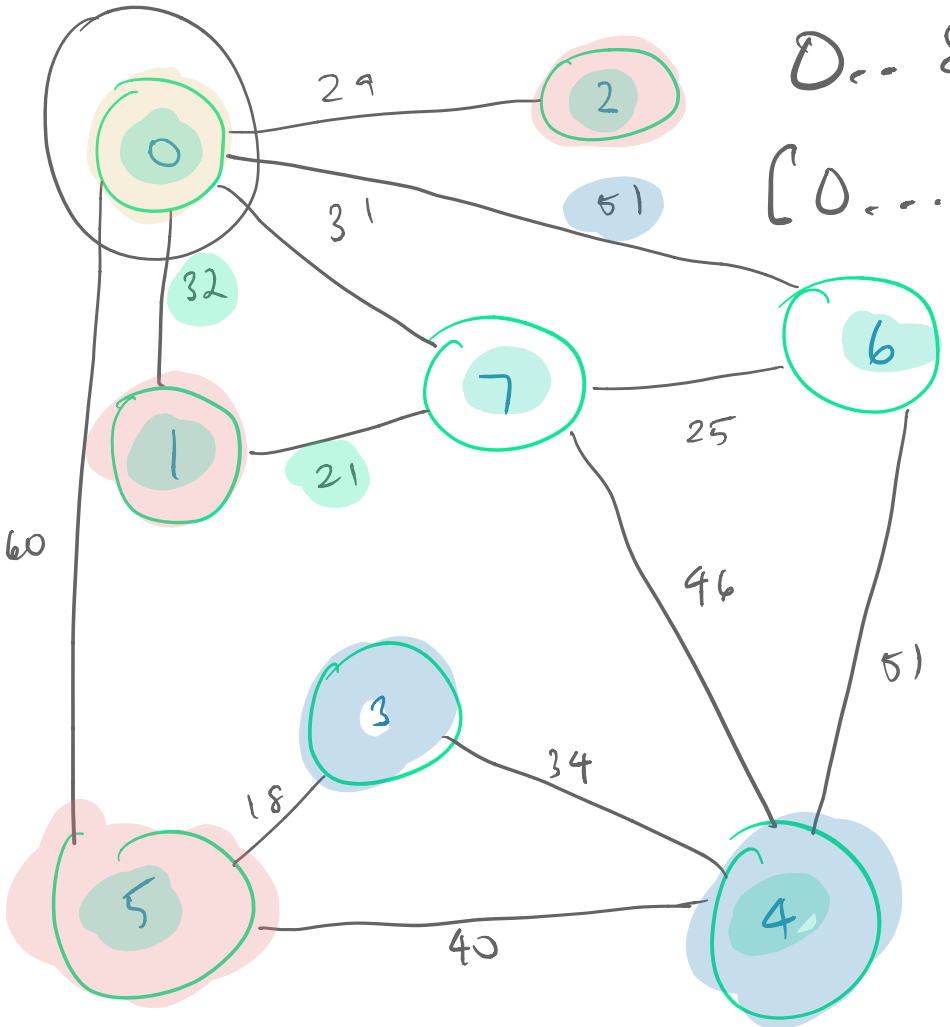
visited :

{ 3 : visited

4 : visited

stdout.

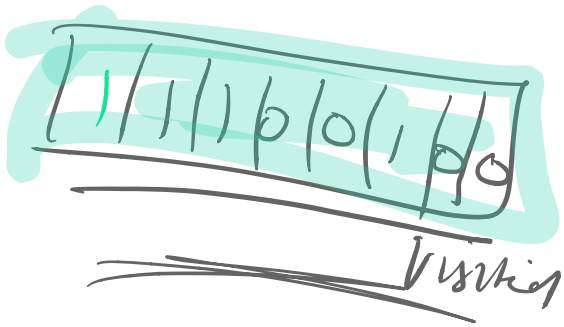
0\n 1\n 2\n


0
2
7
6
1
3
5
4
29
31
51
32
21
60
25
46
51
34
18
40

0... 8

[0...8)


head side
4
3
7
7
6
tail side
queue

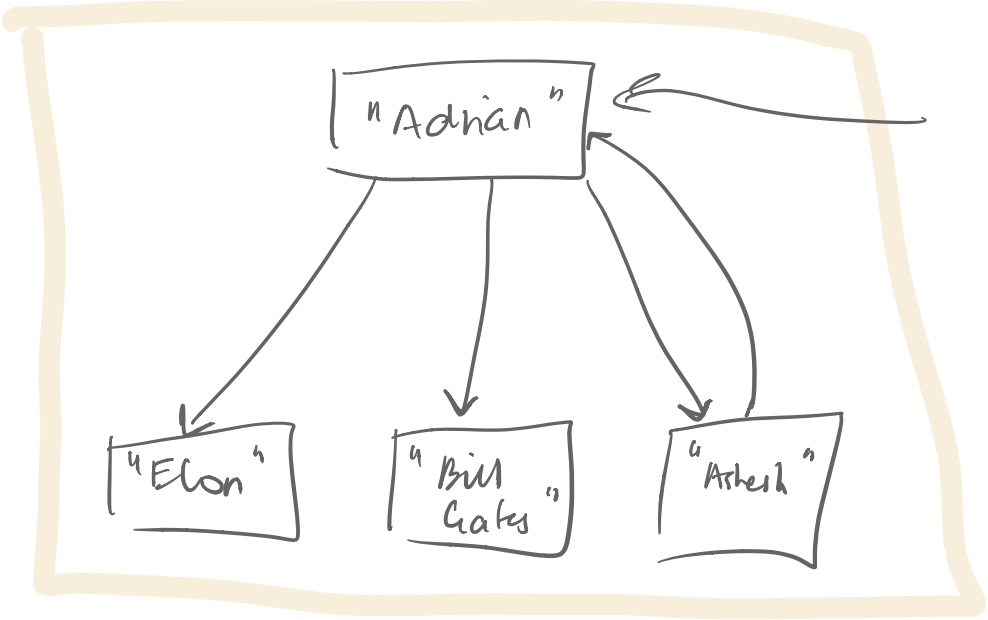

1 1 1 1 0 1 0 1 0 0
visited

x = 5

BFS

VISITED?

We used assumption that all our vertices are labelled with tags from 0..nV.
What if our graph represented a social network?



How would we implement a "visited" test during our DFS/BFS?

Implement function isVisited on graphs with string vertex names.

isVisited :: Char * → "Set" → Bool.

① DIRECT MAP ARRAY. O(1) query.

② LINEAR STRUCTURES.

③ BALANCED TREES! lookup O(log|n|)